

논문 2026-3-3 <http://dx.doi.org/10.29056/jsf.2026.09.03>

HTTP 기반 IoT 장치에서 통신 보안 취약점 분석 및 대처 방법에 관한 연구

염효영*

An Analysis of Security Vulnerabilities and Enhancements for HTTP-based IoT Device Communications

Hyo-Young Yeom*†

요 약

본 연구는 HTTP를 사용하는 IoT 장치의 네트워크 환경에서 발생할 수 있는 보안 위협 중 하나인 패킷 스니핑 공격을 중심으로 보안 취약점을 분석하고 이에 대한 개선 방안을 제시한다. 실험에서는 ESP8266 마이크로컨트롤러를 활용하여 IoT 장치를 구현하고, HTTP 기반 웹 서버에 클라이언트가 접근할 수 있는 통신 환경을 구축하였다. 또한 패킷 스니핑 기법을 이용하여 IoT 장치와 클라이언트 간 통신을 분석함으로써 HTTP 기반 통신에서 발생할 수 있는 보안 취약점을 식별하였다. 이후 제안된 보안 강화 기법을 적용하고 그 효과를 실험적으로 검증하였다. 본 연구는 HTTP 기반 IoT 시스템에서 요구되는 보안 설계의 필요성을 강조하며, 실질적인 보안 강화 방안을 제시한다.

Abstract

This study analyzes security vulnerabilities in IoT device network environments using HTTP, focusing on packet sniffing attacks, and proposes corresponding improvement methods. In the experiment, an IoT device was implemented using an ESP8266 microcontroller, and a communication environment was constructed in which a client could access an HTTP-based web server. In addition, packet sniffing techniques were used to analyze communications between the IoT device and the client to identify security vulnerabilities in HTTP-based communication. The proposed security enhancement method was then applied and its effectiveness was experimentally validated. This study emphasizes the need for secure design in HTTP-based IoT systems and presents a practical approach for improving security.

한글키워드 : IoT보안, 패킷 스니핑, HTTP 취약점, ESP8266, 네트워크 보안

keywords : IoT Security, Packet Sniffing, HTTP Vulnerability, ESP8266, Network Defense

1. 서 론

사물인터넷(IoT)은 스마트홈, 헬스케어, 산업

자동화, 환경 모니터링 등 다양한 분야에서 핵심
기술로 자리를 잡고 있다. IoT 환경에서 활용되
는 통신 프로토콜은 크게 네트워크 접근 계층

* 청주대학교 데이터사이언스학과

† 교신저자: 염효영(email: hyeom@cju.ac.kr)

접수일자: 2026.08.03. 심사완료: 2026.09.04.

게재확정: 2026.09.20.

(Network Access Layer)과 응용 계층 (Application Layer Protocol)으로 구분된다 [1].

네트워크 접근 계층 프로토콜은 물리 계층 및 데이터 링크 계층에서 연결을 담당하며, 전송 거리와 통신 방식에 따라 단거리 무선통신과 장거리 무선통신으로 나뉜다. 단거리 무선통신에는 Wi-Fi와 BLE(Bluetooth Low Energy)가 대표적으로 사용되며, 그 외에도 Zigbee, Z-Wave 등의 프로토콜도 널리 활용된다. 반면, 장거리 무선통신에는 LoRaWAN, NB-IoT, 그리고 이동통신망 기반의 4G/5G 기술이 포함된다 [2].

응용 계층 프로토콜 가운데 경량 메시지 프로토콜, 퍼블리시/서브스크라이브(pub/sub) 방식 사용하는 MQTT(Message Queuing Telemetry Transport)와 REST 기반으로 HTTP와 유사하지만 훨씬 가볍게 설계된 CoAP(Constrained Application Protocol)은 자원 제약형 IoT 환경을 고려하여 경량화된 형태로 설계되었으며[3-4], 메시지 지향 미들웨어용 프로토콜이며 큐 기반 메시지를 전달하는 AMQP(Advanced Message Queuing Protocol)는 MQTT와 CoAP에 비하면 무겁긴 하나 IoT 환경에서 활용되는 대표적인 메시징 프로토콜 중 하나다.[5]

이와 함께 HTTP/HTTPS는 가장 널리 사용되는 웹 기반 응용 계층 프로토콜로서 응용 계층에 속한다. 해당 프로토콜은 구현이 쉽고 범용성이 높다는 장점을 가지지만, 상대적으로 높은 오버헤드로 인해 초기 IoT 장치와 같은 자원 제약형 환경에서는 활용이 제한적이었다. 그러나 최근에는 마이크로컨트롤러의 성능 향상과 가격 하락, 그리고 IoT 시장의 확대에 따라 이러한 자원 제약형 장치에서도 HTTP/HTTPS 기반의 웹 통신 프로토콜 활용이 점차 증가하고 있다 [6-8].

HTTPS는 TLS 기반의 암호화를 통해 높은 수준의 기밀성과 무결성을 제공한다.[9-10] 그러나 TLS 암호화 및 인증 과정은 추가적인 연산과

메모리를 요구하므로, 자원 제약형 MCU 기반 IoT 장치에서는 성능 저하나 메모리 부족 문제가 발생할 수 있다. 또한 인증서 관리와 배포에 따른 운영 비용 및 관리 부담도 고려해야 한다. 이러한 이유로 일부 IoT 시스템에서는 여전히 HTTP 기반 통신이 사용되고 있으며, 이는 평문 전송으로 인해 다양한 보안 위협에 노출될 수 있다. 따라서 HTTP 기반 IoT 통신 환경에서 발생 가능한 보안 취약점을 분석하고 이에 대한 대응 방안을 마련하는 것이 중요하다.

본 연구에서는 패킷 스니핑(Packet Sniffing)을 통해 IoT 장치와 클라이언트 간의 통신을 가로채고 분석하는 공격 시나리오를 설정하여, HTTP 기반 IoT 통신의 보안 취약점을 분석하고 이를 개선하기 위한 경량 보안 기법을 제안하였다. 실험은 ESP8266 마이크로컨트롤러를 활용하여 간단한 IoT 장치를 구현하고, HTTP 기반의 웹서버를 구축하여 클라이언트가 접근할 수 있는 환경에서 수행했다.

2. 연구 방법

2.1 실험용 IoT 장치 설계 및 제작

ESP8266의 GPIO 포트 하나를 원격에서 제어할 수 있도록 구성하였고, AMS2320 온도 습도 센서를 I2C 인터페이스를 통해 연결하였다.

ESP8266에는 4Mbyte 플래시 메모리를 내장하고 있으며, 코어 프로그램뿐 아니라 파일시스템을 구성하면 파일 시스템에 웹 페이지 관련 파일을 저장하여 ESP8266에서 HTTP 서버 기능을 구현하였다. 이 웹서버를 통해서 클라이언트에서 GPIO 스위치를 제어하고 온도 습도 데이터를 읽을 수 있도록 구현하였다.

로그인 페이지는 index.html로 만들었고 센서 데이터, 스위치 상태, 그리고 스위치 제어기능을 display.html에 JavaScript를 이용하여 구현하였

다. 또한 display.html 페이지에 JavaScript를 이용하여 5초 간격으로 센서 데이터를 요청하도록 구현하였다. 클라이언트로 사용한 PC의 IP 주소는 192.168.1.109를 사용하고 ESP8266 웹서버 주소는 192.168.1.12로 지정했다. ESP8266 웹서버와 클라이언트(PC) 사이의 통신 패킷 분석은 네트워크 패킷 분석 도구인 Wireshark를 사용하였다.

패킷 스니핑은 네트워크 인터페이스 카드를 promiscuous mode로 설정해 모든 패킷을 수집한다. 그러나 암호화된 트래픽은 암호를 해독할 능력이 없고 암호화된 형태로 자료를 수집한다.

2.2 보안 단계 1 적용

기본 보안 단계로 사용자 ID와 패스워드로 인증을 통해 IoT 서버에 접근하도록 했다. 이때 클라이언트에서 IoT 서버로 보내는 ID와 패스워드는 암호화하지 않은 상태로 전송하였다. 이 경우 패킷 스니핑을 이용해서 평문 형태의 사용자 인증 정보가 노출되는지를 확인하였다.

2.3 보안 단계 2 적용

클라이언트에서 사용자 ID와 패스워드를, Javascript를 이용해서 AES(Advanced Encryption Standard) 대칭 키 암호화 알고리즘으로 암호화해서 IoT 서버로 보내도록 했다. 이 경우는 패킷 스니핑을 이용해서 패킷을 획득하더라도 암호문만 확인할 수 있으며 사용자 ID와 패스워드의 원문은 확인할 수 없는지를 검증하였다. 또한, 클라이언트가 IoT 서버에서 로그인 html 파일을 받을 때 로그인 페이지에 포함된 JavaScript 코드 및 암호화 키의 노출 가능성을 분석하였다.

2.4 보안 단계 3 적용

클라이언트가 IoT 서버에 접속하면 HTML 및 JavaScript 파일이 클라이언트로 전송된다. 이 과정에서 JavaScript 코드에 포함된 암호화 로직이

나 암호 키가 노출될 가능성이 있으므로, 이를 방지하기 위한 보안 방안을 적용하였다. 그 방법으로 IoT 서버와 통신할 수 있는 독립 실행형 클라이언트를 구현하여 IoT 서버와 API(Application Programming Interface)를 통해 통신하도록 구현하였다. API를 통해 교환되는 모든 명령과 데이터는 AES 대칭키 알고리즘으로 암호화하였다.

AES 암호화를 위해 서버와 클라이언트에 동일한 10개의 AES 키를 사전에 저장하였으며, 데이터 전송 시마다 사용하는 AES 키를 변경하였다. 구현 절차는 다음과 같다. 먼저 클라이언트는 서버에서 사용할 암호화 키를 난수를 이용하여 선택한 후, 서버에 보낼 명령과 함께 기본 키(1번 키)로 암호화하여 전송한다. 명령을 수신한 서버는 이를 복호화한 후 GPIO 포트 제어 명령인 경우 이를 수행한다. 이후 온도·습도 데이터와 GPIO 포트 상태를 수집하여 클라이언트가 전달한 키 번호에 해당하는 AES 키로 암호화한 뒤 클라이언트에 응답한다.

3. 연구 결과

그림 1은 ESP8266 웹서버에서 사용한 index.html 파일로 구현한 로그인 화면이다.

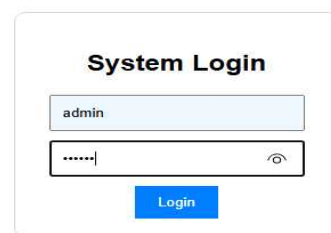


그림 1. 간단한 로그인 화면

Figure 1. Simple login screen

로그인에 성공하면 display.html 페이지가 로

36	2.687366700	192.168.1.12	192.168.1.109	TCP	590 80 → 58316 [ACK] Seq=1073 Ack=631 Win=2050 Len=536 [TCP PDU reassembled in 39]
37	2.687760700	192.168.1.12	192.168.1.109	TCP	590 80 → 58316 [PSH, ACK] Seq=1609 Ack=631 Win=2050 Len=536 [TCP PDU reassembled in 39]
38	2.687801700	192.168.1.109	192.168.1.12	TCP	54 58316 → 80 [ACK] Seq=631 Ack=2145 Win=65535 Len=0
39	2.694775700	192.168.1.12	192.168.1.109	HTTP	311 HTTP/1.1 200 OK (text/html)
40	2.695133200	192.168.1.109	192.168.1.12	TCP	54 58316 → 80 [FIN, ACK] Seq=631 Ack=2402 Win=65278 Len=0
41	2.698509900	192.168.1.12	192.168.1.109	TCP	54 80 → 58316 [FIN, ACK] Seq=2402 Ack=632 Win=2049 Len=0
42	2.698585800	192.168.1.109	192.168.1.12	TCP	54 58316 → 80 [ACK] Seq=632 Ack=2403 Win=65278 Len=0
43	2.735522200	192.168.1.109	203.251.234.8	UDP	71 61886 → 443 Len=29
44	2.737905300	192.168.1.109	192.168.1.12	HTTP	377 GET /readings HTTP/1.1

> Frame 31: Packet, 684 bytes on wire (5472 bits), 684 bytes captured (5472 bits) on interface \Device\NPF{...} > Ethernet II, Src: Intel_88:b8:da (20:16:b9:88:6a:da), Dst: Espressif_a8:d5:44 (e8:db:84:a8:d5:44) > Internet Protocol Version 4, Src: 192.168.1.109, Dst: 192.168.1.12 > Transmission Control Protocol, Src Port: 58316, Dst Port: 80, Seq: 1, Ack: 1, Len: 630 > Hypertext Transfer Protocol > HTML Form URL Encoded: application/x-www-form-urlencoded > Form item: "userid" = "admin" > Form item: "password" = "123456"	0000 e8 db 84 a8 d5 44 20 16 b9 88 6a da 08 00 45 00D... 0010 00 00 e3 7e 40 00 80 06 00 00 c0 a8 01 6d c0 a8 0020 01 0c e3 cc 00 50 e0 04 3d 1c 00 00 5b 8a 50 18P... 0030 ff ff 83 d0 00 00 50 4f 53 54 20 2f 6c 6f 67 69PO ST / 0040 6e 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 n HTTP/1.1 0050 3a 20 31 39 32 2e 31 36 38 2e 31 2e 31 32 0d 0a : 192.16 8.1. 0060 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 Connecti on: 0070 2d 61 6c 69 76 65 0d 0a 43 6f 6e 74 65 6e 74 2d -alive- Cont 0080 4c 65 6e 67 74 68 3a 20 32 38 0d 0a 43 61 63 68 Length: 28-
--	--

그림 3. 보안 1단계에서 로그인 정보 노출과 센서 데이터 노출

Figure 3. At the first security stage, both login information and sensor data are exposed

드되며, 그림 2와 같이 온도·습도 데이터와 LED 상태를 표시하고 LED를 제어할 수 있는 스위치를 제공한다.

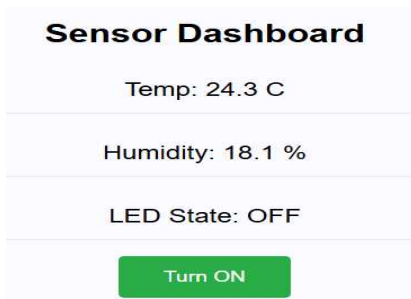


그림 2. 센서 상태를 보여주는 화면

Figure 2. Sensor status display

3.1 보안 1 단계 적용 방법 및 결과

그림 1의 로그인 화면에서 입력한 사용자 ID와 비밀번호를 HTTP POST 방식으로 암호화

없이 서버에 전송하도록 구현하였다.

PC에서 웹 브라우저를 이용하여 ESP8266 웹 서버에 접속하고 로그인하는 과정의 네트워크 트래픽을 Wireshark를 이용하여 분석하였다. 그림 3은 로그인 과정에서 수집한 통신 패킷을 나타낸다. 통신 순서를 살펴보면, PC(192.168.1.109)에서 ESP8266(192.168.1.12)로 로그인 요청이 전송되며(패킷 번호 31), 로그인에 성공하면 서버는 display.html 페이지를 클라이언트로 전송한다(패킷 번호 33~37). 이후 클라이언트는 JavaScript를 통해 센서 데이터 요청을 서버로 전송하고(패킷 번호 44), 서버는 온도, 습도 센서 데이터와 GPIO 상태를 JSON 형식으로 응답한다(패킷 번호 50).

패킷 번호 31을 자세히 살펴보면 사용자 ID가 "admin", 비밀번호가 "123456"로 평문 형태로 전송되는 것을 확인할 수 있다. 따라서 동일한 네트워크에 있는 공격자가 패킷을 수집하면 사용

13	2.057850200	192.168.1.109	192.168.1.12	TCP	54 57438 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
14	2.058354700	192.168.1.109	192.168.1.12	HTTP	469 POST /login HTTP/1.1 (text/plain)
15	2.077733900	192.168.1.12	192.168.1.109	TCP	590 80 → 57438 [ACK] Seq=1 Ack=416 Win=1729 Len=536 [TCP PDU reassembled in 21]
16	2.078949700	192.168.1.12	192.168.1.109	TCP	590 80 → 57438 [PSH, ACK] Seq=537 Ack=416 Win=1729 Len=536 [TCP PDU reassembled in 21]
17	2.078996800	192.168.1.109	192.168.1.12	TCP	54 57438 → 80 [ACK] Seq=416 Ack=1073 Win=65535 Len=0

> Frame 14: Packet, 469 bytes on wire (3752 bits), 469 bytes captured (3752 bits) on interface \Device\NPF{...} > Ethernet II, Src: Intel_88:b8:da (20:16:b9:88:6a:da), Dst: Espressif_a8:d5:44 (e8:db:84:a8:d5:44) > Internet Protocol Version 4, Src: 192.168.1.109, Dst: 192.168.1.12 > Transmission Control Protocol, Src Port: 57438, Dst Port: 80, Seq: 1, Ack: 1, Len: 415 > Hypertext Transfer Protocol > Line-based text data: text/plain > Line: hZBskLq+8bkdlGg==	0000 e8 db 84 a8 d5 44 20 16 b9 88 6a da 08 00 45 00D... 0010 01 c7 fc af 40 00 80 06 00 00 c0 a8 01 6d c0 a8 0020 01 0c e0 5e 00 50 b9 a4 16 f4 00 00 19 73 50 18P... 0030 ff ff 85 83 00 00 50 4f 53 54 20 2f 6c 6f 67 69PO 0040 6e 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 n HTTP/1 0050 3a 20 31 39 32 2e 31 36 38 2e 31 2e 31 32 0d 0a : 192.16 0060 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 Connecti 0070 2d 61 6c 69 76 65 0d 0a 43 6f 6e 74 65 6e 74 2d -alive-
--	---

그림 4. 보안 단계 2에서 서버에 로그인 과정과 인코딩된 로그인 정보

Figure 4. The login process to the server and the encoded login information under security stage 2

1011 13.473472600	192.168.1.109	104.17.24.14	QUIC	86 Protected Payload (KPO), DCID=01e7316f2e79afed45e7136e33799b13eef75049
1012 13.486820900	192.168.1.112	192.168.1.109	TCP	590 80 → 49434 [ACK] Seq=2145 Ack=442 Win=1703 Len=536 [TCP PDU reassembled in 1013]
1013 13.489686600	192.168.1.112	192.168.1.109	HTTP	255 HTTP/1.1 200 OK (text/html)
1014 13.489758500	192.168.1.109	192.168.1.112	TCP	54 49434 → 80 [ACK] Seq=442 Ack=2882 Win=65535 Len=0
1015 13.490466600	192.168.1.109	192.168.1.112	TCP	54 49434 → 80 [FIN, ACK] Seq=442 Ack=2882 Win=65535 Len=0

const userid = document.querySelector("input[name='userid']").value;\n	0000 20 16 b9 88 6a da e8 db 84 a8 d5 44 08 00 45 00	...j...D
const password = document.querySelector("input[name='password']").value;\n	0010 00 f1 00 07 00 00 ff 06 37 36 c0 a8 01 0c c0 a8	...76...
\n\n	0020 01 6d 00 50 c1 1a 00 00 23 e7 c6 a9 9e 07 50 18	...P...#...
const key = CryptoJS.enc.Utf8.parse("1234567890123456"); // 16바이트 키\n	0030 06 a7 18 47 00 00 22 55 73 65 72 20 49 44 22 20	...G...U sen
const iv = CryptoJS.enc.Utf8.parse("1234567890123456"); // 16바이트 IV\n	0040 72 65 71 75 69 72 65 64 3e 0d 0a 20 20 20 20 20	required >...
\n\n	0050 20 20 20 20 20 20 20 20 3c 69 6e 70 75 74 20 74 79	< input
const encrypted = CryptoJS.AES.encrypt(userid + ":" + password, key, {\n	0060 70 65 3d 22 70 61 73 73 77 6f 72 64 22 20 6e 61	pe="pass word
key: key,\n	0070 6d 65 3d 22 70 61 73 73 77 6f 72 64 22 20 70 6c	me="pass word
iv: iv, mode: CryptoJS.mode.CBC, padding: CryptoJS.pad.Pkcs7 }\n	0080 61 65 68 6f 6c 64 65 72 3d 22 50 61 73 73 77	acehold r="P
	0090 6f 72 64 22 20 72 65 71 75 69 72 65 64 3e 0d 0a	ord" req uire
	00a0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	

그림. 5. 보안 2단계에서 index.html 전송 과정 중 보안 키 노출

Figure 5 In Security Stage 2, Security Key Exposure during index.html Transfer

자 인증 정보를 쉽게 획득할 수 있음을 확인하였다.

3.2 보안 2 단계 적용 방법 및 결과

로그인 화면에서 입력한 사용자 ID와 비밀번호를 AES 대칭키로 암호화한 후 HTTP POST 방식으로 서버에 전송하도록 구현하였다.

그림 4는 보안 단계 2를 적용한 후 로그인 과정의 네트워크 패킷을 Wireshark로 분석한 결과를 나타낸다. 패킷을 확인하면 로그인 정보는 암호화된 문자열 "INyHC/hzBskLq+0bkdlGg=="을 확인할 수 있다. 따라서 공격자가 패킷을 가로채더라도 사용자 ID와 비밀번호의 원문을 직접 알 수 없다.

그러나 대칭키 암호화를 클라이언트 측에서 수행하기 위해서는 AES 키와 초기화 벡터(IV)가 JavaScript 코드에 포함되어 클라이언트로 전송

되어야 한다. 이 경우 공격자는 HTML 및 JavaScript 파일을 분석하여 암호화에 사용되는 정보를 획득할 수 있다.

그림 5는 클라이언트가 index.html 파일을 수신하는 과정에서 JavaScript 코드에 포함된 AES 키와 초기화 벡터가 그대로 노출되는 예를 보여준다. 그림 5에서 AES 키 const key = CryptoJS.enc.Utf8.parse("1234567890123456")와 초기화 벡터 const iv = CryptoJS.enc.Utf8.parse("1234567890123456")가 평문으로 포함되어 있음을 확인할 수 있다. 따라서 사용자 인증 정보는 보호할 수 있으나, 암호화에 사용되는 AES 키와 초기화 벡터가 노출되므로 전체 시스템의 보안성은 크게 저하된다.

3.3 보안 3 단계 적용 방법 및 결과

보안 단계 2에서는 클라이언트가 서버로부터

48 4.231482000	192.168.1.109	192.168.1.112	TCP	54 57111 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
49 4.231959600	192.168.1.109	192.168.1.112	HTTP	385 GET /api?cmd=79V27Z%2BLChdvz%2Bhyb608jQ%30%3D HTTP/1.1
50 4.345883800	192.168.1.112	192.168.1.109	TCP	54 80 → 57111 [ACK] Seq=1 Ack=332 Win=1813 Len=0
51 4.375472200	192.168.1.112	192.168.1.109	TCP	249 80 → 57111 [PSH, ACK] Seq=1 Ack=332 Win=1813 Len=195 [TCP PDU reassembled in 53]
52 4.428363300	192.168.1.109	192.168.1.112	TCP	54 57111 → 80 [ACK] Seq=332 Ack=196 Win=65340 Len=0
53 4.431423300	192.168.1.112	192.168.1.109	HTTP	142 HTTP/1.1 200 OK (text/plain)
54 4.477718700	192.168.1.109	192.168.1.112	TCP	54 57111 → 80 [ACK] Seq=332 Ack=284 Win=65252 Len=0

Frame 49: Packet, 385 bytes on wire (3080 bits), 385 bytes captured (3080 bits) on interface \Device\NPF{...} Ethernet II, Src: Intel_88:6a:da (20:16:b9:88:6a:da), Dst: Espressif_a8:d5:44 (e8:db:84:a8:d5:44) Internet Protocol Version 4, Transmission Control Protocol, Hypertext Transfer Protocol	0000 e8 db 84 a8 d5 44 20 16 b9 88 6a da 00 00 45 00	...D...j...E
Request Method: GET	0010 01 73 e6 08 40 00 80 06 00 00 c0 a8 01 6d c0 a8	...@...m...
Request URI: /api?cmd=79V27Z%2BLChdvz%2Bhyb608jQ%30%3D	0020 01 0c df 17 00 50 29 50 f3 57 00 00 1a 47 50 18	...P...W...GP
Request URI Path: /api	0030 ff ff 85 2f 00 00 47 45 54 20 ff 61 70 69 3f 63	...GE T /api?
Request URI Query: cmd=79V27Z%2BLChdvz%2Bhyb608jQ%30%3D	0040 6d 64 3d 37 39 56 32 37 5a 25 32 42 4c 43 68 64	...md=79V27 Z%2BLChdvz%2Bhyb60 8jQ%30%3D
Request Version: HTTP/1.1	0050 56 7a 4d 68 79 62 36 4f 38 6a 51 25 33 44 25 33	... HTTP/1.1...Host
Host: 192.168.1.12	0060 3a 20 31 39 32 2e 31 36 38 2e 31 2e 31 32 0d 0a	...: 192.16 8.1.12...
	0070 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70	...Connecti on: keep
	0080 2d 61 6c 69 76 65 0d 0a 55 73 65 72 2d 41 67 65	...-alive... User-Agent
	0090 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20	...nt: Mozilla/5.0
	00a0 28 57 69 6e 64 6f 77 73 20 4e 54 20 31 30 2e 30	... (Windows NT 10.0
	00b0 3b 20 57 69 6e 36 34 3b 20 78 36 34 29 20 41 70	...; Win64; x64) Ap

그림 6. 보안 단계 3에서 클라이언트에서 서버로 명령어 전송

Figure 6. Client-to-server command transfer under Security Stage 3

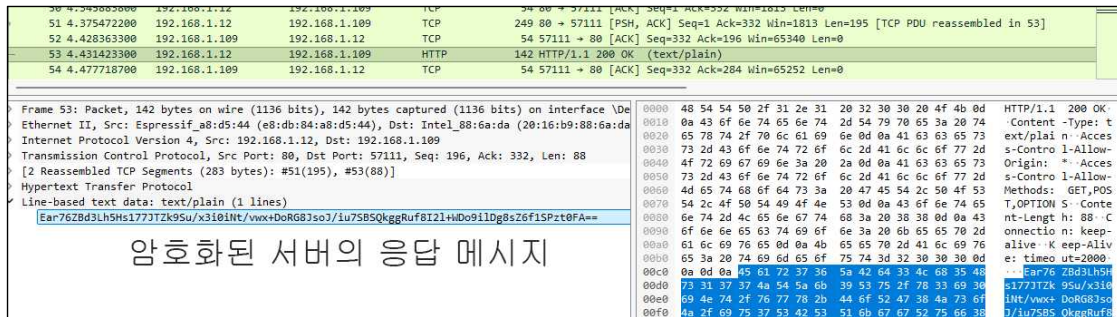


그림 7 보안 단계 3에서 서버에서 클라이언트로 자료 전송
Figure 7. Data transmission from server to client under Security Stage 3

HTML 및 JavaScript 파일을 다운로드하는 과정에서 AES 키와 초기화 벡터가 노출되는 취약점이 존재한다. 따라서 이번 단계에서는 클라이언트 프로그램을 독립 실행형 응용프로그램으로 구현하고, 서버는 미리 정의된 API를 통해서만 접근을 허용하였다.

AES 암호화를 위해 128비트 AES 키 10개를 사전에 생성하여 서버와 클라이언트에 동일하게 저장하였다. 1번 AES 키는 요청 메시지를 보호하기 위한 기본 키로 사용하였으며, 응답 메시지는 요청에 포함된 키 번호에 해당하는 AES 키를 사용하여 암호화하였다.

클라이언트는 서버로 데이터를 요청할 때 요청 명령과 서버 응답에 사용할 키 번호를 하나의 문자열로 구성한 후 1번 AES 키를 이용하여 암호화하여 전송하였다. 서버는 수신한 문자열을 1번 AES 키로 복호화한 후 명령을 수행한다. 이

후 센서 데이터와 GPIO 상태를 수집하고, 요청 메시지에 포함된 키 번호에 해당하는 AES 키로 암호화하여 클라이언트에 응답하였다. 클라이언트는 해당 키 번호에 대응하는 AES 키를 이용하여 데이터를 복호화한 후 결과를 표시하였다.

그림 6은 클라이언트가 서버로 데이터 요청을 전송하는 과정에서 Wireshark로 통신 패킷 수집 결과를 나타낸다. 그림 7은 서버가 클라이언트의 요청에 응답하는 패킷을 나타낸다.

그림 6에서 클라이언트가 서버로 전송한 명령은 GET/api?cmd=79V27Z%2BLChdVzMhyb608jQ%3D%3D HTTP/1.1로 확인되며, AES로 암호화된 문자열만 관찰할 수 있다. 서버에서 이를 복호화하면 readData&key=1로 나타나며, readData는 온도, 습도 및 GPIO 상태를 요청하는 명령이고 key=1은 서버가 응답 메시지를 암호화할 때 1번 AES 키를 사용하도록 지정하는 값이다.

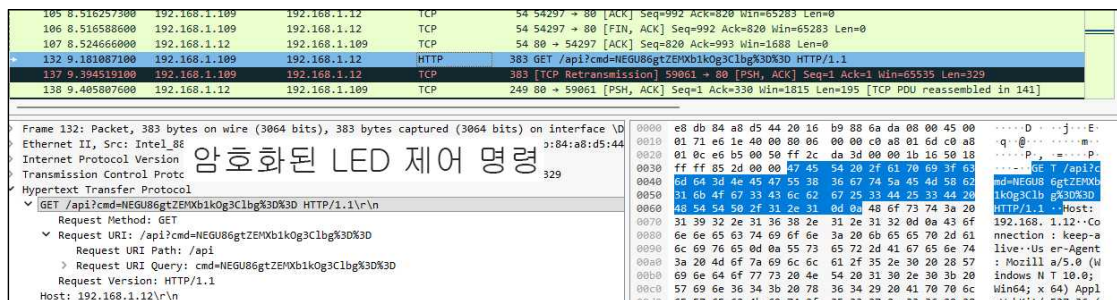


그림 8. 보안 단계 3에서 클라이언트에서 서버로 LED 명령 전송
Figure 8. Client-to-server LED command transfer under Security Stage 3

이에 대응하는 서버의 응답 메시지는 그림 7과 같이 Ear76ZBd3Lh5Hs177JTzk9Su/x3i0iNt/vwx+DoRG8JsoJ/iu7SBSQkggRuf8I2l+WDo9ilDg8sZ6f1SPzt0FA== 로 전송된다. 클라이언트는 미리 공유된 1번 AES 키를 이용하여 이를 복호화하며, 결과적으로 {"temperature": "29.2", "humidity": "38.6", "iostate": "OFF"} 데이터를 확인할 수 있다.

LED 제어 명령의 경우 그림 8과 같이 GET/api?cmd=NEGU86gtZEMXb1kOg3Clbg%3D%3D HTTP/1.1로 전송되며, 이를 복호화하면 led&key=6으로 해석된다. 여기서 led는 LED 상태를 토글(toggle) 하는 명령으로, 현재 상태가 OFF이면 ON으로, ON이면 OFF로 변경된다.

이 명령에 대한 서버 응답은 그림 9에 있으며, 응답 데이터는 Z1wdq3VBr0A9g5InYjRK9pMIEgUSei5qsW6JjZAtbs6rtjN6ncBNs3Hl9swoa6vUT018G9wGJjDcROSffgNuIA==이다. 이를 복호화하면 {"temperature": "29.2", "humidity": "38.6", "iostate": "ON"}으로 나타나며, LED 상태가 ON으로 변경되었음을 확인할 수 있다.

표 1에 각 보안 단계별 로그인 소요시간과 명령 실행 소요시간을 측정해서 기록했다. 서버 코드에 로그인 부분과 명령 실행 부분에 micros() 명령을 사용해서 마이크로초(10^{-6} 초) 까지 12회 측정해서 평균값을 기록했다.

표 1 보안 단계별 서버의 로그인 및 명령 실행 소요 시간

Table 1. Login and command execution time at each security stage on the server

	로그인(ms)	명령실행(ms)
보안1단계	0.073	227.8
보안2단계	0.650	227.5
보안3단계	1.163	235.2
추가인증	1.161	237.3

4. 분석 및 제안

4.1 보안 1 단계

보안 단계 1에서는 암호화되지 않은 로그인 정보가 네트워크 패킷을 통해 그대로 노출되어, 통신 과정 패킷을 수집하면 사용자 인증 정보가 쉽게 유출되는 등 보안상 매우 취약함을 확인하였다.

4.2 보안 2 단계

보안 단계 2에서는 로그인 정보를 AES로 암호화함으로써 패킷을 수집해도 암호화된 문자열만 확인되며, 원문 로그인 정보는 확인할 수 없음을 확인하였다. 그러나 클라이언트가 IoT 웹서버에 접속하는 과정에서 서버로부터 HTML 및 JavaScript 파일을 다운로드하게 되며, 이 과정에서 암호화 키 및 초기화 벡터가 클라이언트 측

140 9.452218200	192.168.1.109	192.168.1.12	TCP	54 59061 → 80 [ACK] Seq=330 Ack=196 Win=65340 Len=0
141 9.454549600	192.168.1.12	192.168.1.109	HTTP	142 HTTP/1.1 200 OK (text/plain)
142 9.496175700	192.168.1.109	192.168.1.12	TCP	54 59061 → 80 [ACK] Seq=330 Ack=284 Win=65252 Len=0
164 11.392495800	192.168.1.12	192.168.1.109	TCP	54 80 → 59061 [FIN, ACK] Seq=284 Ack=330 Win=1815 Len=0
165 11.392574200	192.168.1.109	192.168.1.12	TCP	54 59061 → 80 [ACK] Seq=330 Ack=285 Win=65252 Len=0
180 12.825607500	192.168.1.109	192.168.1.12	TCP	54 59061 → 80 [FIN, ACK] Seq=330 Ack=285 Win=65252 Len=0

Frame 141: Packet, 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface \Device\NPF{...} Ethernet II, Src: Espressif_a8:d5:44 (e8:db:84:a8:d5:44), Dst: Intel_88:6a:da (20:16:b9:88:6a:da) Internet Protocol v4 Transmission Control Protocol [2 Reassembled TCP segments, Seq=54549600-54549600] Hypertext Transfer Protocol Line-based text data: text/plain (1 lines) Z1wdq3VBr0A9g5InYjRK9pMIEgUSei5qsW6JjZAtbs6rtjN6ncBNs3Hl9swoa6vUT018G9wGJjDcROSffgNuIA==	0000 48 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d HTTP/1.1 200 OK 0010 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 Content-Type: text/plain 0020 65 78 74 2f 70 6c 61 69 6e 0d 0a 41 63 63 65 73 ext/plain 0030 73 2d 43 6f 6e 74 72 6f 6c 2d 41 6c 6c 6f 77 2d s-Control-Allow- 0040 4f 72 69 67 69 6e 3a 20 2a 0d 0a 41 63 63 65 73 Origin: * 0050 73 2d 43 6f 6e 74 72 6f 6c 2d 41 6c 6c 6f 77 2d s-Control-Allow- 0060 4d 65 74 68 6f 64 73 3a 20 47 45 54 2c 50 4f 53 Methods: GET, POST, OPTIONS 0070 54 2c 4f 50 54 49 4f 4e 53 0d 0a 43 6f 6e 74 65 Content-Type: text/plain 0080 6e 74 2d 4c 65 6e 67 74 68 3a 20 38 0d 0a 43 Content-Length: 88 0090 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d Connection: keep-alive
--	--

그림 9 보안 단계 3에서 서버에서 클라이언트로 자료 전송
Figure 9. Data transmission from server to client under Security Stage 3

코드에 포함되어 노출되는 문제점이 발생하였다.

4.3 보안 3 단계

보안 단계 3에서는 이러한 문제를 해결하기 위해 IoT 서버 접근을 API 기반으로 제한하고, 클라이언트 프로그램은 서버로부터 HTML 및 JavaScript 코드를 직접 전달받지 않고 미리 저장된 파일을 실행하도록 구성하였다. 다만, 저장된 파일(HTML, JavaScript)로 API에 접근하려면 로컬 웹서버가 필요하다. 파이썬을 사용하는 경우 내장 웹서버를 활용하거나 경량의 공개 웹서버 프로그램을 사용할 수 있다. 또한 보안상의 이유로 로컬 파일에서 직접 웹서버를 실행할 수 없는데, 이는 ESP8266 서버에서 필요에 따라 웹 브라우저가 다른 도메인의 리소스에 접근할 수 있도록 허용하는 메커니즘인 CORS(Cross - Origin Resource Sharing) 설정을 통해 해결할 수 있다.

이 방식은 IoT 장치가 일반적으로 불특정 다수가 접근하는 웹 서비스가 아니라 제한된 클라이언트 환경에서 동작한다는 특성을 고려한 것으로, 보안 단계 2에서 발생한 키 노출 문제를 근본적으로 해결할 수 있다.

한편 IoT 장치는 대부분 제어기능을 포함하므로, 공격자가 암호화된 명령 문자열을 가로챌 후 다시 서버로 재전송하는 경우 서버가 이를 정상 명령으로 인식하여 실행할 가능성이 존재한다. 이는 단순한 데이터 유출을 넘어 장치 오동작 및 제어권 탈취로 이어질 수 있는 심각한 보안 위협이다.

4.4 제어 명령 추가 인증

이를 해결하기 위해 본 연구에서는 추가적인 인증 절차를 제안한다. 서버와 클라이언트에는 동일한 1에서 100까지 번호와 이에 대응하는 4자리 난수 값 100쌍을 미리 저장하고, 제어 명령이 전송되면 서버는 클라이언트로 1에서 100 사이의

번호를 난수 발생으로 생성 후 전달한다. 클라이언트는 받은 번호에 대응하는 저장된 4자리 값을 서버로 전송하며, 서버는 수신된 값이 저장된 값과 일치할 때만 제어 명령을 실행하도록 구성한다. 이와 같이 키 기반 암호화와 더불어 요청-응답 기반의 추가 인증 절차를 적용함으로써, 단순 패킷 재전송 공격 및 제어 명령 위조 공격에 대한 보안성을 향상할 수 있다.

저자의 연구실에서 운영하는 서버에 백업용 HDD를 제어 명령 추가 인증 방법을 사용하여 물리적 연결을 제어해서 사용 중이다.

4.5 보안 단계별 처리 시간 분석

표 1의 로그인 소요 시간을 비교하면, 1단계와 2단계 사이에서 약 8.9배의 시간 차이가 발생하는 것을 확인할 수 있다. 이는 2단계에서 적용된 암호화 과정에 추가적인 처리 시간이 소요되기 때문으로 판단된다. 또한 2단계와 3단계에서도 약 1.8배의 차이가 나타나는데, 이는 HTTP를 통해 직접 명령을 전달하는 방식보다 Web API를 통해 명령을 처리하는 과정에서 추가적인 처리 시간이 발생한 것으로 분석된다.

명령 실행 시간의 경우, 암호화를 적용하지 않은 1단계와 추가 인증을 적용한 3단계를 비교하면 약 10 ms의 차이가 발생한다. 이는 전체 명령 실행 시간의 약 4.4%에 해당하는 수준이다. 따라서 추가 인증과 같은 보안 기능을 적용하더라도 명령 실행 과정에서 발생하는 보안 처리 시간은 전체 처리 시간에서 차지하는 비중이 상대적으로 작으며, 특히 센서로부터 데이터를 읽어오는 시간에 비해 매우 작은 수준임을 확인할 수 있다.

5. 결론

본 연구에서는 ESP8266 기반 IoT 환경에서 HTTP 기반 통신 구조의 보안 강화를 위해 API 기반 독립 실행형 클라이언트 구조와 AES 기반

다중 키 암호화 방식을 적용하였다. 또한 요청 메시지에 키 번호를 포함하여 응답 시 동적으로 암호화 키를 선택하도록 설계함으로써 패킷 스니핑 환경에서도 데이터가 암호문 형태로만 노출되도록 하였다. 더불어 제어 명령에 대한 추가 인증 절차를 적용하여 단순 재전송 공격 및 무단 제어 가능성을 줄일 수 있음을 확인하였다.

참 고 문 헌

- [1] V. Karagiannis, F. Vázquez-Gallego, P. Chatzimisios and J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things," *Ad Hoc Networks*, vol. 35, pp. 104–124, 2015.
DOI: 10.5281/zenodo.51613
- [2] R. K. Shilpa and H. K. Saini, "Comparative Analysis of Wireless Communication Technologies for IoT Applications," *International Journal of Engineering and Applied Sciences (IJEAS)*, vol. 9, no. 1, pp. 23–31, 2022.
<http://www.ijeas.org/comparative-analysis-of-wireless-communication-technologies-for-iot-applications>
- [3] OASIS Standard, MQTT Version 3.1.1, 2014.
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>
- [4] A. Banks and R. Gupta, "MQTT Version 3.1.1 Specification", OASIS Standard, 2014.
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>
- [5] J. Rellermeyer et al., "The AMQP Protocol: Efficient Messaging for Distributed Systems,"
<https://ieeexplore.ieee.org/document/7470538>
- [6] Nugraha, I. R., Putra, W. H. N., & Setiawan, E. (2024). A Comparative Study of HTTP and MQTT for IoT Applications in Hydroponics. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, 8(1), 119 – 126. DOI:10.29207/resti.v8i1.5561
- [7] Dauda, A., Flauzac, O., & Nolot, F. (2024). A Survey on IoT Application Architectures. *Sensors*, 24(16), 5320.
DOI: 10.3390/s24165320
- [8] Hong, S., Kang, J., & Kwon, Y. (2023). Performance comparison of HTTP, HTTPS, and MQTT for IoT applications. *Journal of the Korea Institute of Information and Communication Engineering*, 27(5), 611 – 618.
DOI: 10.7236/IJASC.2023.12.1.9
- [9] E. Rescorla, RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3, IETF, 2018.
<https://www.rfc-editor.org/rfc/rfc8446>
- [10] R. Fielding et al., RFC 9110: HTTP Semantics, IETF, 2022.
<https://www.rfc-editor.org/rfc/rfc9110>

저 자 소 개



염효영(Hyo-Young Yeom)

1986.2 단국대 물리학과 학사
1988.2 단국대 물리학과 석사
1994.8 단국대 물리학과 박사
2000.2–2006.8 Brown University Division of Engineering Research Associate.
2020–현재 청주대학교 데이터사이언스학과 교수
<주관심분야> 마이크로컨트롤러, 임베디드, IOT, 데이터베이스, 계측 장비 설계 및 제작