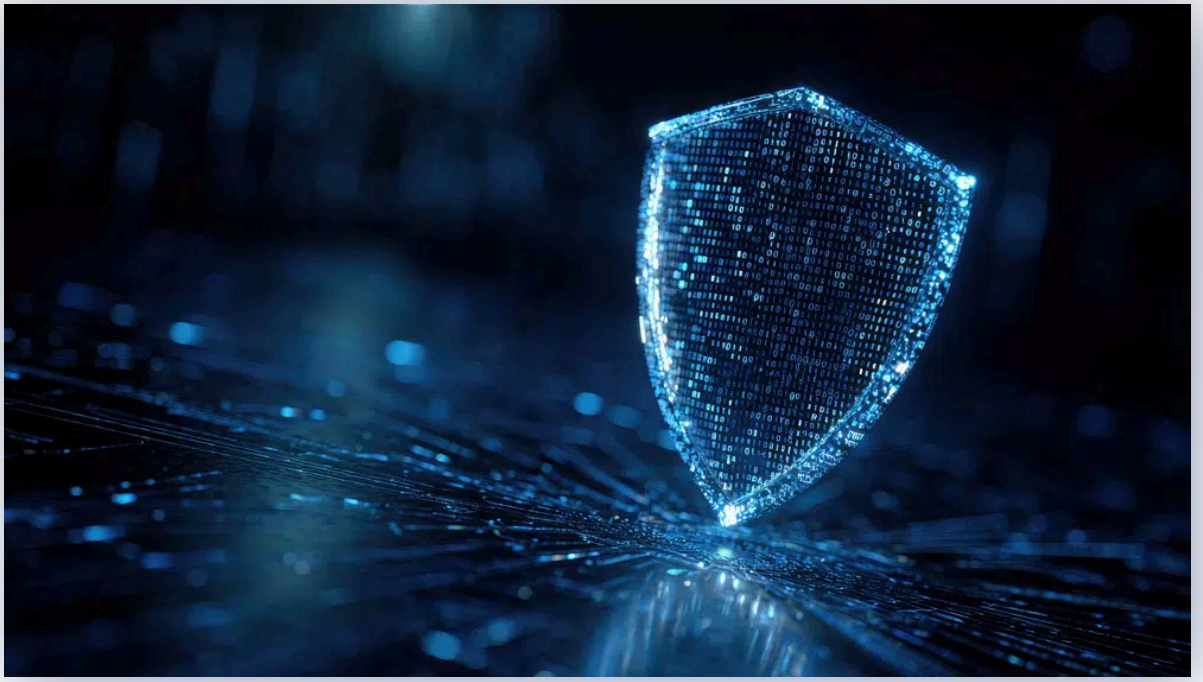
Breaking Down the Advantech WISE-DeviceOn CVE-2025-34256 Vulnerability



April 15, 2026

Cybersecurity

By: Alex Williams

Unlocking the full potential of IoT requires more than powerful hardware—it demands strong security. Advantech, a global
n $2 billion in annual revenue, powers industries like smart
industrial PCs, embedded systems, and edge computing platforms.
n Internet of Things (IoT) management platform that enables
ts through a web application.

nsibly disclosed 11 vulnerabilities discovered in Advantech WISE-
osed in December 2025. This article breaks down the most critical

exchange system that can be used as an alternative to standard cookies. JWTs use the following format:

eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.

eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ.

dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk

JWTs consists of three sections separated by periods: header, payload, and signature. Each section is Base64-encoded. The previous JWT can be decoded into the following:

```
{
  "typ": "JWT",
  "alg": "HS256"
}
{
  "iss": "joe",
  "exp": 1300819380,
  "http://example.com/is_root": true
}
{
    // Signature
}
```

More information on JWTs can be found in the JSON Web Token open standard, RFC 7519.

When a user attempts to log in to DeviceOn, the client issues an HTTP GET request to the endpoint "/rmm/v1/accounts/login". This request includes an *Authentication Basic* HTTP header that contains the attempted username and password. If the credentials are valid, then the server responds by issuing a signed JWT. This token is returned to the client and stored in the HTTP cookie *ElRMMToken*. The JWT is subsequently used to authenticate and authorize the user for future requests. An example DeviceOn JWT token is shown below:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJuYW1lIjoicm9vdCIsImV4cCI6MTc1ODgzNjk3NCwibG9naW4
iOiIyMDI1LTA5LTI1IDE1OjQ5OjM0ICgtMDUwMCkiLCJpYXQiOjE3NTg4MzMzNzQsImVtYWlsIjoicm9vdEBhZHZ
hbnRlY2guY29tLnR3In0.x17OI5xPWSUe6HNU5B7W5U6eam5rYH2Iaj9CeFPwJD5_FcGDi8ForP2vwC0DQZhw0dVA7fSoMrBK7X,

Which can be Base64-decoded into the following sections:

```
    "exp": 1758836974,
    "login": "2025-09-25 15:49:34 (-0500)",
    "iat": 1758833374,
    "email": "root@advantech.com.tw"
}
{
    // Signature
}
```

The name of the logged in account is stored in the value of the JSON key *name,* and the email associated with the account is stored in the value of the JSON key *email*. Note that when a user logs into the web application, the email is used as the account's username.

JWT EIRMMToken Use of Hard-coded Cryptographic Key (CVE- 2025-34256)

Our researchers found that the following Java class, *server\portal\WEB-INF\lib\RMMLib.jar!com.utils.AuthUtil*, is responsible for the authentication and authorization process:

```
public class AuthUtil {
    private static final Logger a = LogManager.getLogger(AuthUtil.class);
    public static final MultiLanguage language = new MultiLanguage();
    public static final String CN_BUILTIN = "EIRMMToken";
    public static final String CN_NAME = "EIName";
    public static final String CN_SSO = "EIToken";
    private static final String b = "2FA-Session-ID";
    private static final String c = "Basic ";
    private static final String d = "Bearer ";
    private static final String e = "Authorization";
    private static final String f = "X-DeviceOn-Auth";
    private static final String g = "X-DeviceOn-LDAP";
    private static final String h = "X-DeviceOn-2FA";
    private static final String i = "JWT@EI-RMM@WISE-PaaS";

// [...Truncated...]
```

```
IError.ServerError.getCode(), "server is running into a JWT problem"));
        }
        this.t = JWT.require((Algorithm)this.s).build();
        this.r = sScheme.equalsIgnoreCase("https");
        if (this.b()) {
            switch (method) {
                case "basic": {
                    if (authorization.isEmpty()) {
                        throw new APIException(APIResponse.fail(Response.Status.F
ORBIDDEN, APIError.AuthNotProvidedError.getCode(), "no Basic data in
Authorization header"));
                    }
                    this.a(authorization);
                    break;
                }
                case "bearer": {
                    if (authorization.isEmpty()) {
                        throw new APIException(APIResponse.fail(Response.Status.F
ORBIDDEN, APIError.AuthNotProvidedError.getCode(), "no Bearer data in
Authorization header"));
                    }
                    g.SSOTokenPreValidation(authorization, false);
                    if (p.processSRP(this) == null) break;
                    this.v = g.decodeAuthFromSSOToken(authorization, null, false)
;
                    break;
                }
                case "eirmmtoken": {
                    this.decodeAuthFromBuiltin(authorization);

// [...Truncated...]

    public void decodeAuthFromBuiltin(String eiToken) throws APIException {
        Account account;
        DecodedJWT jwt = this.b(eiToken);
        Claim email = this.a(jwt, "email");
        try {
            account = this.u.getAccountByMail(email.asString());
        }
        catch (Exception e) {
            a.error((Object)e);
            throw new APIException(APIResponse.fail(Response.Status.FORBIDDEN, AP
IError.AuthError.getCode(), "no such user for this token"));
        }
        if (account != null) {
            this.v = new Auth(account, eiToken, CN_BUILTIN, false);
        }
    }
```

During our research we discovered that WISE-DeviceOn uses a hardcoded key to sign the secret for every JWT (this was confirmed on another installation of the product on a different machine). The hardcoded key used is “JWT@EI-

```
= "JWT@EI-RMM@WISE-PaaS"; // [1]
```

thm [2] and the verifier of the JWT [3]:

```
IError.ServerError.getCode(), "server is running into a JWT problem"));
        }
        this.t = JWT.require((Algorithm)this.s).build(); // [3]
```

This can be seen later after the login process was successful that the same algorithm is used to sign the user's new JWT [4]:

```
    private String a(Map<String, String> strClaims) throws APIException {
        try {
            HashMap<String, Object> header = new HashMap<String, Object>();
            header.put("alg", this.s.getName());
            long now = System.currentTimeMillis();
            long timeout2 = (long)SystemConfig.getInstance().getLoginSetting().ge
tIdleTimeout() * 60000L;
            JWTCreator.Builder builder = JWT.create().withHeader(header).withIssu
edAt(new Date(now)).withExpiresAt(new Date(now + timeout2));
            for (Map.Entry<String, String> e : strClaims.entrySet()) {
                builder.withClaim(e.getKey(), e.getValue());
            }
            return builder.sign(this.s); // [4]
```

When a user sends an HTTP request with the JWT *EIRMMToken* included in the HTTP *Cookie* header, the following function is used to verify the JWT using the verifier that was built using the hardcoded key [5]:

```
    private DecodedJWT b(String token) throws APIException {
        try {
            return this.t.verify(token); // [5]
        }
        catch (JWTVerificationException e) {
            if (e instanceof TokenExpiredException) {
                throw new APIException(APIResponse.fail(Response.Status.UNAUTHORI
ZED, APIError.AuthTokenExpiredError.getCode()));
            }
            a.error(e);
            throw new APIException(APIResponse.fail(Response.Status.FORBIDDEN, AP
IError.AuthTokenVerificationError.getCode()));
        }
```

After the JWT is successfully verified, the *email* claim's value in the JWT is retrieved [6] and evaluated to ensure the associated email address is for a valid account [7]. If it is, then the user is successfully authenticated [8]:

```
        }
        catch (Exception e) {
            a.error(e);
            throw new APIException(APIResponse.fail(Response.Status.FORBIDDEN, API
IError.AuthError.getCode(), "no such user for this token"));
        }
        if (account != null) {
            this.v = new Auth(account, eiToken, CN_BUILTIN, false); // [8]
```

The only JWT field that is ensured to be valid is the JWT claim *email field*. As a result, if an attacker can determine the email address associated with a victim's account, they can forge a valid JWT by signing it with the hardcoded secret key. This allows the attacker to impersonate the victim and perform any actions permitted to that account.

An additional factor that significantly reduces the attack complexity is the format requirement for the Super Administrator account. The email address associated with this account must begin with the prefix *root@*, which substantially narrows the search space for brute-force or guess-based attacks.

For example, a JWT forged for a Super Administrator account with the email address *root@advantech.com.tw* is accepted by the application: *eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJlbWFpbCI6InJvb3RAYWR2YW50ZWNoLmNvbS50dyJ9.Qmxiuuv5iWF85NAd3MFbHAL1iUelN92-tELt8FCpfptnR_GEkZqVk3lLAM19NlP0UidZv0eD59h0OMj42mLmoQ*

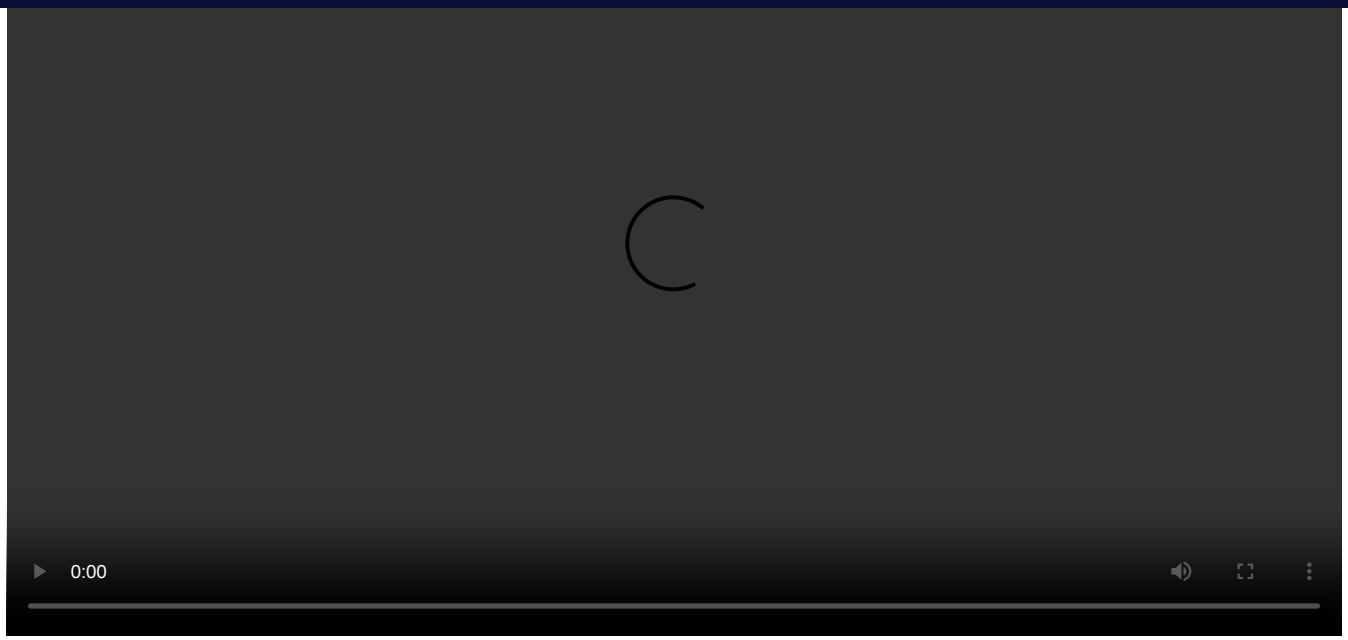
Which can be Base64-decoded to the following:

```
{
    "typ": "JWT",
    "alg": "HS512"
}
{
    "email": "root@advantech.com.tw"
}
{
    // Signature using secret "JWT@EI-RMM@WISE-PaaS"
}
```

After Bypassing Authentication

If the attacker can guess, or brute force, the email associated with the "Super admin" account (which again is the email that must begin with "root@") and create a JWT signed with the hardcoded key, the attacker can abuse the following capabilities:

- covery locations (which can be remote servers), server certificates,
- gent's device via the *Device > Remote Control > Launch*
- nt devices by uploading malicious apps via the *Application >*



Click play to view a video of the proof of concept.

The proof of concept above performs the following actions:

- Creates and signs a JWT using the hardcoded secret.
- Sends an HTTP POST request to upload a malicious application which will be installed on all connected Windows agents.
- Sends an HTTP GET request to retrieve the list of connected Windows agents.
- Sends an HTTP POST request to install and execute the previously uploaded malicious application to all connected Windows agents.
- This malicious application will execute the commands "*whoami > C:\RCE_OUTPUT.txt*" and "*calc.exe*".

Mitigations

Advantech has released an advisory addressing these vulnerabilities. They have also released a patched version that fixes these vulnerabilities.

Timeline

- **October 10, 2025** – Pellera reports the 11 vulnerabilities found to VulnCheck to work as an intermediary between Pellera and Advantech. VulnCheck responds that Advantech has been contacted about the vulnerabilities.
- **October 21, 2025** – VulnCheck states that Advantech has requested the PoC files (which could not be included in the initial report to VulnCheck). Pellera sends the PoCs to Advantech via OneDrive.
nCheck, additional information about the vulnerabilities and cross-site

heck, a cloud instance running the patched version to ensure that the that all vulnerabilities are confirmed to be patched.
g CVEs to the vulnerabilities: CVE-2025-34256, CVE-2025-34257, 260, CVE-2025-34261, CVE-2025-34262, CVE-2025-34263, CVE-266.

Understand your exposure before it gets exploited. Get in touch to learn how our assessments can strengthen your security posture.

Talk to a Security Expert

Alex Williams is an Offensive Security Researcher at Pellera.

← Previous Blog

Follow Us

Recent Posts

Unleashing the Power of a 3D Data Security Architecture

Identity Is the New Perimeter: The 5 Must Haves for Securing AI Access Control

Turning the Promise of AI in Cybersecurity into Real Customer Value

Want To Read More?

Visit Blog

Unleashing the Power of a 3D Data Security Architecture

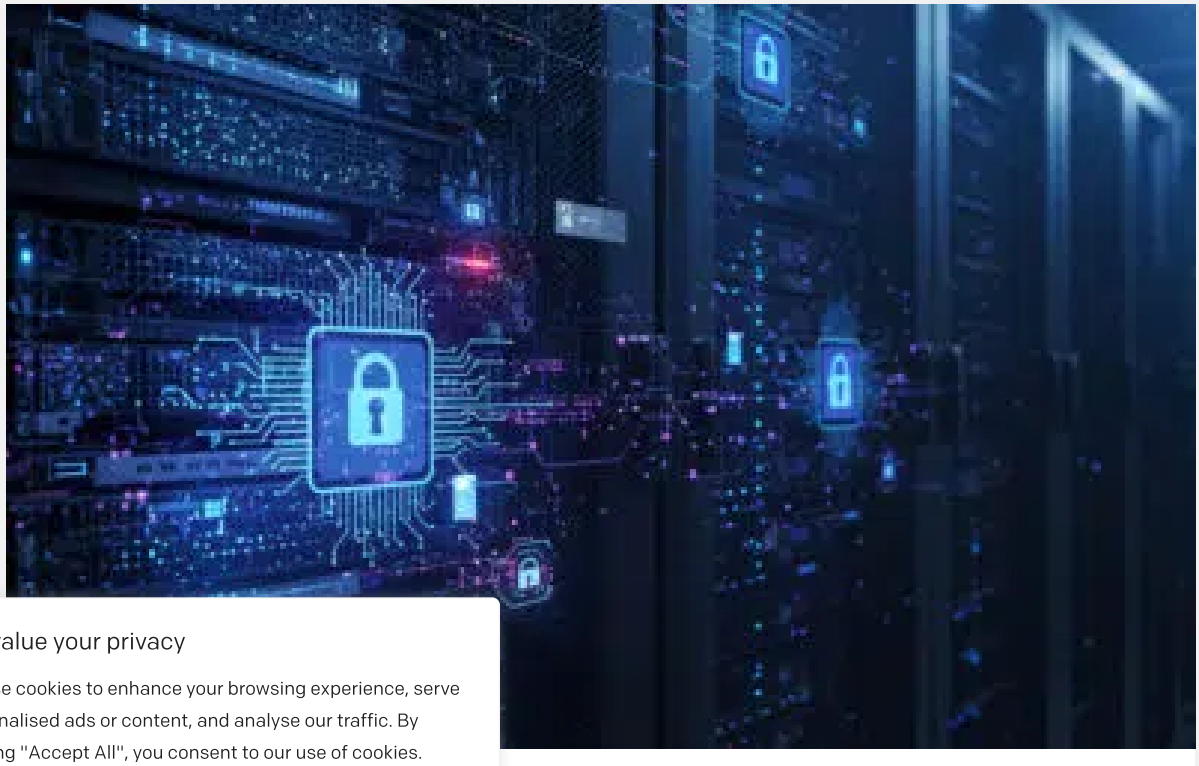
by Pellera Technologies

By: Chris Dedham Relying on traditional security measures is no longer enough. For years, organizations have built...

Identity Is the New Perimeter: The 5 Must Haves for Securing AI Access Control

by Pellera Technologies

By: David Santeramo Artificial intelligence is rapidly reshaping how organizations operate. By implementing strong AI...

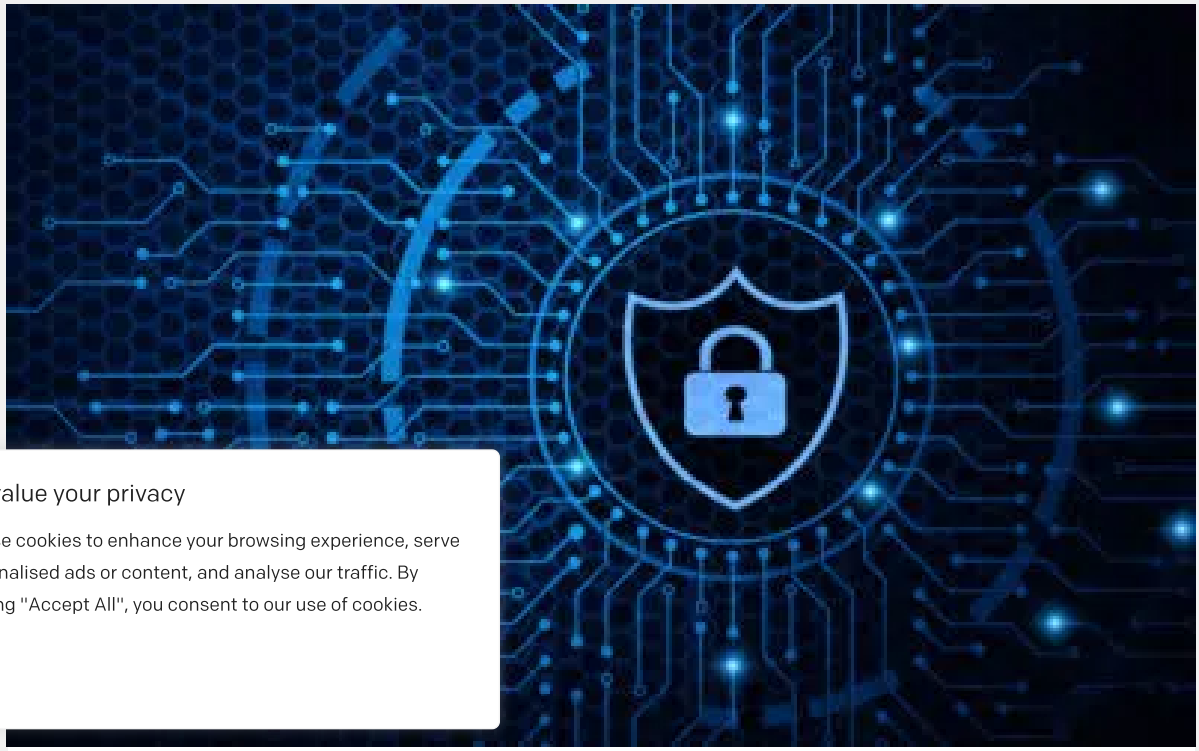


into Real Customer Value

Unleashing the Power of a 3D Data Security Architecture

by Pellera Technologies

By: Chris Dedham Relying on traditional security measures is no longer enough. For years, organizations have built...



Visit Blog Page

Momentum builds here

Company

About Us

AIM

Leadership

Contact Us

Insights

Blog

News

Podcast

Get in Touch

Join Our Team

Converge Technology Solutions

Mainline Information Systems