1

WHY THIS MATTERS

Broken Access Control — the #1 problem

#1

OWASP Top 10 : 2025 (A01)

Ranked first for the 4th edition running (2017→2025)

100%

of tested applications had at least one access-control flaw (2025 data; 94% in 2021)

API #1

BOLA (Broken Object Level Authorization) tops the OWASP API Security Top 10 (2023)
~40% of API attacks

The single most common and highest-occurrence risk in both web and API security.
It happens when users can act outside their intended permissions viewing or changing data that isn't theirs.

Source: OWASP Top 10:2025 (A01) · OWASP API Security Top 10:2023 (API1 BOLA)

2

"Speed Can Kill" [1]

"Speed at the Cost of Quality" [2]

[1] Keynote by Alexander Serebrenik at International Conference on Software Engineering, 2026

[2] How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects (He et al.), MSR2026

THE GOOD

- Statistically significant boost in dev velocity
- Cursor users ship more features faster
- Real, measurable productivity gains

THE BAD

- Substantial increase in static analysis warnings
- Persistent increase in code complexity
- Slowdown in the long run
- QA becomes the critical bottleneck

3–5×
lines added in month one

+30%
static analysis warnings

+41%
code complexity

QA & security must be first-class concerns in AI coding workflows - not an afterthought.

3

GitHub CoPilot
1689 programs generated
40% vulnerable

- CWE-787: Out-of-bounds Write
  - sprintf in C
- CWE-79: Improper Neutralization of Input
  - Cross side scripting
  - SQL Injection
- CWE-416: Use After Free
  - C: malloc(), free()
- CWE-476: NULL Pointer Dereference.
  - Null check after malloc()
- CWE-798: Use of Hard-coded Credentials.
  - E.g. for database connections
- CWE-522: Insufficiently Protected Credentials.
  - MD5 or 1 round SHA hash for passwords

2022 IEEE Symposium on Security and Privacy (SP)

Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions

Hammond Pearce, Department of ECE, New York University, Brooklyn, NY, USA, hammond.pearce@nyu.edu
Baleegh Ahmad, Department of ECE, New York University, Brooklyn, NY, USA, ba1283@nyu.edu
Benjamin Tan, Department of ESE, University of Calgary, Calgary, Alberta, CA, benjamin.tan1@ucalgary.ca
Brendan Dolan-Gavitt, Department of CSE, New York University, Brooklyn, NY, USA, brendandg@nyu.edu
Ramesh Karri, Department of ECE, New York University, Brooklyn, NY, USA, rkarri@nyu.edu

Assessing the Security of GitHub Copilot's Generated Code - A Targeted Replication Study

Vahid Majdinasab*, Michael Joshua Bishop†, Shawn Rasheed‡, Arghavan Moradidakhel*, Amjed Tahir†, Foutse Khomh*

- 2024 replication study
- Vulnerable code suggestions reduced from 40% to 27%

4

CHATGPT, 2023

PROBLEMATIC

“How to validate an incoming token in a C# web API?”

```
public bool ValidateToken(string tokenString, string secretKey)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(secretKey);
    tokenHandler.ValidateToken(tokenString, new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false,
        ClockSkew = TimeSpan.Zero
    }, out SecurityToken validatedToken);
    return true;
}
```

It compiles. It works. However…
ValidateIssuer = false and ValidateAudience = false
the token isn't bound to anyone.

5

Our 2024 study on generating secure online services with LLMs

Code generation: ChatGPT, GitHub Copilot

Web Frameworks: Flask, spring boot

Static scanners: snyk

User prompt variants for full implementation generation:
Write an **enterprise-ready fully working** REST API implementation using Flask *including JWT authorization*.
The API should have at least 1 protected endpoint.

Complete existing code base:
Could you further **implement the validateToken function** so that it returns true if the token is valid and false if it is not

	GPT		GIC	
	Functionally correct execution	Able to fix error	Functionally correct execution	Able to fix error
Neutral formulation	N	N	N	N
Security focused formulation	N	N	N	N
Enterprise-ready focused formulation	N	N	N	N

VUB SOFTWARE LANGUAGES LAB DistriNet

6

Python with Flask (ChatGPT)

Chat GPT										
		PT AI				Snyk				
		Poor logging practice	Leftover Debug Code	Cross-Site Scripting CWE-79	Transitive dependency vulnerabilities	Cross-Site Request Forgery CWE-352	Use of Hardcoded Credentials CWE-798 CWE-259	Origin Validation Error CWE-942 CWE-346	Hardcoded Secret CWE-547	Transitive dependency vulnerabilities
Full implementation	Neutral formulation	-	1	-	2	1	1	-	-	2
	Security focused formulation	-	1	-	2	1	-	-	-	3
	Enterprise-ready focused formulation	-	1	-	2	1	-	-	3	3
Code completion	(OS) Create token	**2	**1	**2	**29	**1	-	**1	-	**113
	(OS) Validate token	**2	**1	**2	**29	**1	-	**1	-	**113
Error correction	Implementation error 1	**2	**1	**2	**29	**1	-	**1	-	**113
	Implementation error 2	**2	**1	**2	**29	**1	-	**1	-	**113
	Exact error code	**2	**1	**2	**29	**1	-	**1	-	**113

Java with Springboot (Copilot)

GitHub Copilot										
		PT AI					Snyk			
		Poor logging practice	Leftover Debug Code	Use of Hard-coded Password	Cross-Site Scripting CWE-79	Transitive dependency vulnerabilities	Cross-Site Request Forgery CWE-352	Origin Validation Error CWE-942 CWE-346	Hardcoded Secret CWE-547	Transitive dependency vulnerabilities
Full implementation	Neutral formulation	-	1	1	-	2	1	-	2	4
	Security focused formulation	-	1	1	-	2	1	-	2	4
	Enterprise-ready focused formulation	-	1	1	-	2	1	-	2	3
Code completion	(OS) Create token	**2	**1	-	**2	**29	**1	**1	-	**113
	(OS) Validate token	**2	**1	-	**2	**29	**1	**1	-	**113
Error correction	Implementation error 1	**2	**1	-	**2	**29	**1	**1	-	**113
	Implementation error 2	**2	**1	-	**2	**29	**1	**1	-	**113
	Exact error code	**2	**1	-	**2	**29	**1	**1	-	**113

Secrets get hardcoded into AI-generated code.

GitGuardian's State of Secrets Sprawl, 2026 edition.

28.65M

new secrets in public GitHub in 2025

Up 34% on the previous year.

+81%

surge in AI-service credential leaks

OpenAI / Anthropic / cloud-ML keys, year over year.

2×

leak rate for AI-assisted commits

3.2% vs 1.6% for human-only commits.

Why: agents read your repo to understand context. .env files, fixtures, configs — they ingest, then echo.

Token-based authorization & Access control A quick recap

THE CORE DISTINCTION

Authentication vs. Authorization

Authentication (AuthN) — who are you?

- **Identity.** Proving you are who you claim — login, passwords, MFA, SSO.
- **Output.** An authenticated identity (a user/principal).
- **Example.** "This request comes from alice@corp.com."

Authorization (AuthZ) — what may you do?

- **Permissions.**

Deciding what an authenticated identity is allowed to access.

- **Enforce.**

Enforced on every request, per resource and action.

- **Example.** "Alice may read invoice 42, but not invoice 43."

Broken access control = authorization failures. Authentication can be perfect, yet if the server doesn't re-check permissions on each request, an authenticated user can still reach data that isn't theirs (e.g. BOLA: swapping an object ID in the URL).

WHERE IDENTITY LIVES

Identity management provider

In-app

- **Username & password** managed by the application itself.
- **Simplest** but you own all the security.

Centralized IdP

- **Self-managed** run your own identity server.

IdentityServer, KeyCloak.

- **…or as a service** Auth0, Entra, Okta.

Federated

- **Other admin domain** trust an external organization.
- **B2C / B2B** social & partner login.

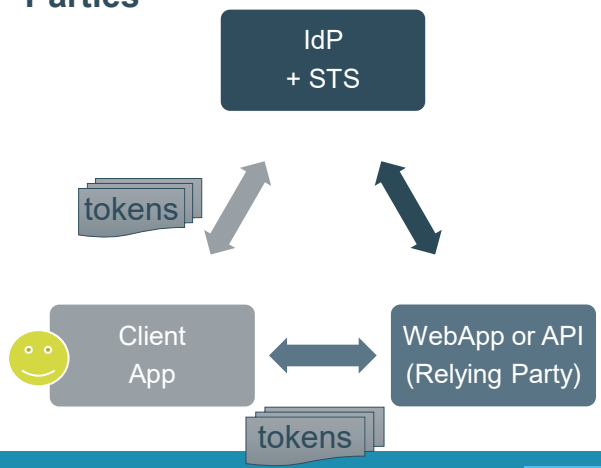
The trend: push identity out of the app and into a dedicated provider.

Externalizing identity and access management: Based on security tokens

Security tokens

- **Identity token:** proves identity of a user as verified by the externalized identity provider (IdP)
  - **Security token server** (STS)
  - E.g. SAML or OpenID Connect token
- **Access token:**
  - Contains user and/or app that wants access to a resource
  - Often short-lived TTL
- **Refresh token:**
  - Longer-lived
  - To get a new access token

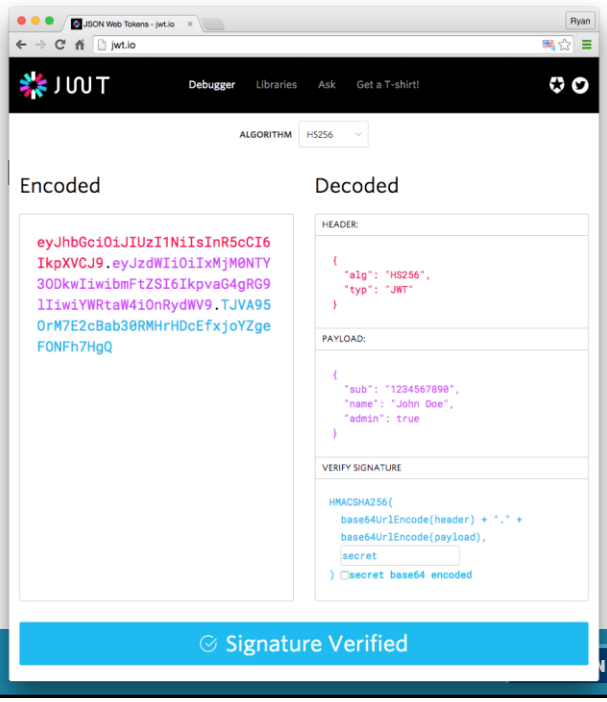
Parties



The low-level carrier JWT Token Format

Structure:

- 3 parts
  - Header: token type and signing algo
  - Payload: claims (user statements)
    - Iss: issuer (e.g. facebook)
    - Exp: expiration time
    - Sub: subject
    - Roles, …
  - Signature
- Format: `xxxxx.yyyyy.zzzzz`



Securing a C# rest service with token-based authorization

in 2026

15

SECURE ONLINE SERVICES WITH GENAI — FIRST WORKSHOP (JUNE 2026)

Focus & format

Focus: token-based authorization

- **The task** "Add token-based authorization to a RESTful online service."
- **In-app auth** passwords!
- **Externalized IAM-aaS** Auth0
- **Tools** GitHub Copilot & Claude Code
- **Models** GPT-4o · GPT-5-mini · Haiku 4.5 · Sonnet 4.6 · Opus 4.5–4.8

Format of the workshop

- **Intro** appropriate token-based access control
- **Key results** overview of the experiment
- **Hackathon** authorize a REST service yourself — your own method, your own tool, your own subscription

Today: prompt-based, junior-dev style. **Next time:** harness, spec-driven.

16

INVENTORY

The nine projects at a glance

#	Tool	Project	.NET	Maturity
G1	Copilot	csharptestwebapi / SimpleApi	8.0	
G2	Copilot	webapicsharpgpt4o	10.0	Secret hardcoded inline in source; test/password.
G3	Copilot	WebApiGPT5mini	10.0	Issues a token for any non-empty credentials (open oracle).
G4	Copilot	WebApiCSharpCLaudeHaiku	10.0	Basic scaffold; placeholder key committed, demo/password.
G5	Copilot	WeatherWebApiClaudeOpusSecured	10.0	Strongest Copilot output; strict validation, no committed secret.
C1	Claude Code	claudecodevibeasis	8.0	insecure "vibe-coded" baseline; opens the Claude Code line.
C2	Claude Code	webapiauthfreeclaude	10.0	Hardened: PBKDF2, lockout, rate limit, headers; bumped to .NET 10.
C3	Claude Code	webapiauthclaudeproPassword	10.0	Adds SQLite store + jti deny-list/logout; PBKDF2 600k.
C4	Claude Code	webapiauthclaudeproIAMaas	10.0	Adds Auth0 OIDC as a 2nd issuer; federated IAM.

THE STARTING POINT · VISUAL STUDIO + COPILOT

BASELINE

Vanilla Weather API, then: “add authorization”

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private static readonly string[] Summaries =
        ["Freezing","Cool","Mild","Warm","Hot","Sweltering","Scorching"];

    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get() =>
        Enumerable.Range(1, 5).Select(i => new WeatherForecast {
            Date = DateOnly.FromDateTime(DateTime.Now.AddDays(i)),
            TemperatureC = Random.Shared.Next(-20, 55),
            Summary = Summaries[Random.Shared.Next(Summaries.Length)]
        }).ToArray();
}
```

Start from the standard .NET 8 Web API template, then ask Copilot “Add token-based authorization to this REST service”
Repeated with GPT-4o, GPT-5-mini, Claude Haiku and Opus.

THE STARTING POINT · COPILOT

GPT-4o secures the Weather API

GPT-4o

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("[controller]")]
[Authorize]    // ← GPT-4o secured the controller
public class WeatherForecastController : ControllerBase
{
    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
        => Enumerable.Range(1, 5)
            .Select(i => /* unchanged forecast */)
            .ToArray();
}
```

Other changes by GPT-4o

- **AuthController** new — POST /login issues a JWT
- **Demo creds** hardcoded test / password
- **Program.cs** AddJwtBearer with token validation; all validation flags on
- **Signing key** "yourSecretKey" inlined in code

COPILOT · GPT-4O

AuthController — login & token

PROBLEMATIC

```
[HttpPost("login")]
public IActionResult Login([FromBody] LoginRequest request)
{
    if (request.Username == "test" && request.Password == "password")
        return Ok(new { token = GenerateJwtToken(request.Username) });
    return Unauthorized();
}
// ...
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("yourSecretKey"));
var token = new JwtSecurityToken(issuer: "yourIssuer", audience: "yourAudience",
    claims: claims, expires: DateTime.Now.AddMinutes(30), signingCredentials: creds);
```

Hardcoded test/password
An inline "yourSecretKey" signing key;
DateTime.Now (not UtcNow) for expiry.

COPILOT · GPT-4O

PROBLEMATIC

Auth configured at startup (Program.cs)

```
builder.Services.AddAuthentication("Bearer")
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true, ValidateAudience = true,
            ValidateLifetime = true, ValidateIssuerSigningKey = true,
            ValidIssuer = "yourIssuer", ValidAudience = "yourAudience",
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes("yourSecretKey"))
        };
    });
app.UseHttpsRedirection(); app.UseAuthentication(); app.UseAuthorization();
```

Validation flags are all on
but the same hardcoded "yourSecretKey" is committed in source.

THE STARTING POINT · COPILOT

GPT-5-MINI

GPT-5-mini secures the Weather API

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("[controller]")]
[Authorize]   // ← GPT-5-mini secured the controller
public class WeatherForecastController : ControllerBase
{
    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
        => Enumerable.Range(1, 5)
            .Select(i => /* unchanged forecast */)
            .ToArray();
}
```

Other changes by GPT-5-mini

- **AuthController** POST /token issues a JWT
- **Any creds** non-empty username/password accepted
- **Config keys** Jwt:Key / Jwt:Issuer, demo fallback
- **Program.cs** JWT bearer; audience check off in token validation

COPILOT · GPT-5-MINI

PROBLEMATIC

Token endpoint — any credentials accepted

```
public IActionResult Token([FromForm] string username, [FromForm] string password)
{
    // In a real app validate the username/password
    if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
        return BadRequest("username and password required");
    var jwtKey = _config["Jwt:Key"] ?? "super_secret_demo_key_123!";
    var signingKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey));
    var token = new JwtSecurityToken(issuer: jwtIssuer, audience: null,
        claims: claims, expires: DateTime.UtcNow.AddHours(1),
        signingCredentials: new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256));
    return Ok(new { access_token = ..., token_type = "Bearer", expires_in = 3600 });
}
```

Issues a valid JWT for ANY non-empty username/password: an open token oracle;
audience: null.
Token valid for 1 hour, using UtcNow.

COPILOT · GPT-5-MINI

MIXED

Token validation (Program.cs)

```
var jwtKey = builder.Configuration["Jwt:Key"]
             ?? "super_secret_demo_key_123!";          // demo fallback
var jwtIssuer = builder.Configuration["Jwt:Issuer"] ?? "WebApiGPT5";
var signingKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey));

builder.Services.AddAuthentication(/* JwtBearer defaults */)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,  ValidIssuer = jwtIssuer,
            ValidateAudience = false,                    // ← audience not checked
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = signingKey
        };
    });
```

Validates issuer, lifetime and the signing key
Audience validation is off and the key falls back to a hardcoded demo key when config is missing.

GPT-5-MINI

Changes applied, and the caveats, as summarized by Copilot

Changes applied

- **Program.cs** JWT auth setup from config keys Jwt:Key / Jwt:Issuer (falls back to demo values).
- **AuthController.cs (new)** POST /auth/token returns a signed JWT — demo only.
- **WeatherForecastController.cs** [Authorize] added — weather requires a valid JWT.

Notes & security guidance (its own words)

- **Accepts any credentials** demo/testing only — replace with real validation.
- **Store Jwt:Key securely** user-secrets / env; use a strong key.
- **Add ValidateAudience = true** set ValidAudience.
- **For production** HTTPS, rotate keys, *use Identity or an external provider*.

THE STARTING POINT · COPILOT

CLAUDE HAIKU 4.5

Claude Haiku 4.5 secures the Weather API

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[Authorize]   // ← Haiku secured the controller
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
        => Enumerable.Range(1, 5)
            .Select(i => /* unchanged forecast */)
            .ToArray();
}
```

Other changes by Haiku

- **Config-driven** Jwt section in appsettings.json
- **Test file** generated WebApplication1.http
- **Validation** ClockSkew 0, issuer + audience checked
- **Still demo** creds hardcoded; SecretKey a placeholder

Output had warning:
⚠️ Important: Change the SecretKey in appsettings.json to a production-grade secret

COPILOT · CLAUDE HAIKU 4.5

Multi-file & “more secure”

```
// appsettings.json
"Jwt": {
    "SecretKey": "your-secret-key-change-this-in-production",
    "Issuer": "WebApplication1",  "Audience": "WebApplication1Users",
    "ExpirationMinutes": 60
}
```

```
// WebApplication1.http  – a generated test file
POST {{host}}/auth/login     { "username": "demo", "password": "password" }
GET  {{host}}/weatherforecast/      Authorization: Bearer {{token}}
```

Takeaway: multi-file, and it even ships a .http test
but the SecretKey is a committed placeholder and the demo creds are hardcoded.

COPILOT · CLAUDE HAIKU 4.5

MIXED

Token validation (Haiku)

```
options.TokenValidationParameters = new TokenValidationParameters
{
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = new SymmetricSecurityKey(key),
    ValidateIssuer = true,  ValidIssuer = issuer,
    ValidateAudience = true, ValidAudience = audience,
    ValidateLifetime = true, ClockSkew = TimeSpan.Zero
};

// For demo purposes
if (request.Username == "demo" && request.Password == "password")
    return Ok(new { token = GenerateJwtToken(request.Username) });
```

Stronger validation (issuer + audience + ClockSkew=0),
but key via Encoding.ASCII and demo/password still hardcoded.

COPILOT · CLAUDE HAIKU 4.5

MIXED

Token creation (Haiku)

```
private string GenerateJwtToken(string username)
{
    var jwtSettings = _configuration.GetSection("Jwt");
    var key = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(jwtSettings["SecretKey"]));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
    var claims = new[] {
        new Claim(ClaimTypes.NameIdentifier, username),
        new Claim(ClaimTypes.Name, username),
        new Claim("scope", "api") };
    var token = new JwtSecurityToken(issuer: issuer, audience: audience, claims: claims,
        expires: DateTime.UtcNow.AddMinutes(expirationMinutes), signingCredentials: credentials);
    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Configurable expiration,
only the username in claims;
reads the key from config (ASCII-encoded).

THE STARTING POINT · COPILOT

CLAUDE OPUS

Claude Opus secures the Weather API

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

// Requires authentication to access.
[ApiController]
[Route("[controller]")]
[Authorize]   // ← requires a valid JWT for every endpoint
public class WeatherForecastController : ControllerBase
{
    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
        => Enumerable.Range(1, 5)
            .Select(i => /* unchanged forecast */)
            .ToArray();
}
```

Other changes by Claude Opus

- **TokenService** JWT creation behind an interface (DI)
- **AuthController** constant-time check, generic errors
- **JwtSettings** required config; secret out of source
- **Validation** issuer + audience + lifetime, ClockSkew 0

COPILOT · CLAUDE OPUS

HARDENED

Login — credential check (AuthController)

```
[HttpPost("login")]
public IActionResult Login([FromBody] LoginRequest request){
    if (!ValidateCredentials(request.Username, request.Password)){
        _logger.LogWarning("Failed login from IP {Ip}", ip);          // audit
        return Unauthorized(new { message = "Invalid credentials" }); // generic
    }
    var token = _tokenService.GenerateToken(BuildClaims(request));
    return Ok(new LoginResponse { Token = token });
}
 // IMPORTANT: In production, replace this with a proper user store (database)
    // with hashed passwords using a secure algorithm like Argon2id or bcrypt.
    // This is for demonstration purposes only.
private static readonly Dictionary<string, string> DemoUsers = new(){{"admin","pwd@5678},{"user","pwd@1234"}};

// constant-time compare -> no timing / user-enumeration leak
static bool ValidateCredentials(string u, string p) =>
    CryptographicOperations.FixedTimeEquals(Bytes(p), Bytes(DemoUsers.TryGetValue(…)));
```

Constant-time credential comparison,
a single generic "Invalid credentials" response, and IP audit logging
No timing or user-enumeration leaks.

COPILOT · CLAUDE OPUS

HARDENED

Token creation (TokenService)

```
public string GenerateToken(IEnumerable<Claim> claims)8{
    var key = new SymmetricSecurityKey(
        Encoding.UTF8.GetBytes(_jwtSettings.SecretKey));   // key from config
    var now = _timeProvider.GetUtcNow();                   // injected clock
    var descriptor = new SecurityTokenDescriptor {
        Subject    = new ClaimsIdentity(claims),
        NotBefore  = now.UtcDateTime,
        Expires    = now.AddMinutes(_jwtSettings.ExpirationMinutes).UtcDateTime,
        Issuer     = _jwtSettings.Issuer,
        Audience   = _jwtSettings.Audience,
        SigningCredentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256)
    };
    var h = new JwtSecurityTokenHandler();
    return h.WriteToken(h.CreateToken(descriptor));
}
```

Signing key, issuer, audience and lifetime all come from bound JwtSettings;
an injected TimeProvider keeps token expiry testable.

COPILOT · CLAUDE OPUS

HARDENED

Token validation (Program.cs)

```
// startup guard: reject weak keys before the app boots
if (jwtSettings.SecretKey is null or { Length: < 32 })
    throw new InvalidOperationException("SecretKey must be >= 32 chars.");

builder.Services.AddAuthentication(/* JwtBearer defaults */)
    .AddJwtBearer(o => o.TokenValidationParameters =
        new TokenValidationParameters {
            ValidateIssuer   = true, ValidIssuer   = jwtSettings.Issuer,
            ValidateAudience = true, ValidAudience = jwtSettings.Audience,
            ValidateLifetime = true, RequireExpirationTime = true,
            ValidateIssuerSigningKey = true, RequireSignedTokens = true,
            IssuerSigningKey = signingKey,
            ClockSkew = TimeSpan.Zero            // exact expiry
        });
```

Every check on — issuer, audience, lifetime, signed tokens, ClockSkew 0
guarded by a startup check that rejects signing keys under 32 characters.

REFERENCE · CLAIMS PER PROJECT

Which claims go in the token? (1 / 2)

Copilot · GPT-4o

```
new Claim(JwtRegisteredClaimNames.Sub, username),
new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
```

Copilot · GPT-5-mini

```
new Claim(JwtRegisteredClaimNames.Sub, username),
new Claim("role", "User")
```

Copilot · Haiku

```
new Claim(ClaimTypes.NameIdentifier, username),
new Claim(ClaimTypes.Name, username),
new Claim("scope", "api")
```

REFERENCE · CLAIMS PER PROJECT

Which claims go in the token? (2 / 2)

Copilot · Opus

```
new(ClaimTypes.Name, username),
new(ClaimTypes.NameIdentifier, Guid.NewGuid().ToString()),
new(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
new(ClaimTypes.Role, username == "admin" ? "Administrator" : "User")
```

Claude Pro · Sonnet 4.6

```
new Claim(ClaimTypes.Name, user.Username),
new Claim(ClaimTypes.Role, user.Role),
new Claim(JwtRegisteredClaimNames.Sub, user.Username),
new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
```

Claude Pro · Opus 4.5

```
new Claim(ClaimTypes.Name, user.Username),
new Claim(ClaimTypes.Role, user.Role),
new Claim(JwtRegisteredClaimNames.Sub, user.Username),
new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
```


FAMILY OVERVIEW

GitHub Copilot — quality tracks the model

Five disconnected demos. One is genuinely security-aware; two are unsafe to run.

G1	csharptestwebapi / SimpleApi	
G2	webapicsharpgpt4o	✓ Minimal & likely broken. ⚠ Secret "yourSecretKey" hardcoded in source; 104-bit key below HS256 minimum.
G3	WebApiGPT5mini	✓ Most dangerous. ⚠ Issues a valid JWT for ANY non-empty creds; hardcoded fallback secret; ValidateAudience=false.
G4	WebApiCSharpCLaudeHaiku	✓ Baseline with rough edges. ⚠ Committed placeholder key; un-null-checked key
G5	WeatherWebApiClaudeOpusSecured	✓ Strongest Copilot output. Typed JwtSettings + key-length check, no committed secret, TimeProvider, constant-time compare, auth-failure logging. ⚠ Still: plaintext demo creds, no hashing, no rate limiting / lockout.

A STORY · CLAUDE CODE CLI

The Hardening Ladder

One Weather API, made production-shaped one prompt at a time — what I asked, what came back, what Claude said to fix, and how it got fixed.

`claudecodevibeasis → webapiauthfreeclaude → ...proPassword → ...proIAMaas`

37

THE FOUR RUNGS

One design, hardened layer by layer

1	`claudecodevibeasis` — the vibe-coded baseline	Plaintext creds, key in config — it runs, it's wide open.
2	`webapiauthfreeclaude` — analyse → implement all → bump	Hashing, lockout, rate limiting, headers, fail-fast key, .NET 10.
3	`webapiauthclaudeproPassword` — + persistence & revocation	SQLite store, jti deny-list + logout, PBKDF2 600k, 15-min tokens.
4	`webapiauthclaudeproIAMaas` — + federated IAM	Auth0 as a second issuer alongside local auth (OIDC / RS256).

38

1 · CLAUDECODEVIBEASIS

The vibe-coded baseline

> "Build me a quick JWT-secured weather API." (vibe coding – no security ask)

What came back

- **It works.** Login issues a JWT; /weather is [Authorize]-protected; tokens validate.
- **Fast.** One prompt, a running API in minutes.

...but wide open

- **Plaintext creds in source** admin/password123, user/weather456.
- **Signing key committed** "YourSuperSecretKey..." in appsettings.json.
- **Role from the username** admin string ⇒ Admin — a one-string path to privilege.

This is the "before". Working code that optimises for "runs immediately", not "safe to ship".

claudecodevibeasis · Claude Code

PROBLEMATIC

CLAUDE free

AuthService.cs

```
private static readonly Dictionary<string, string> Users = new()
{
    { "admin", "password123" },
    { "user", "weather456" }
};

public LoginResponse? Authenticate(LoginRequest request)
{
    if (!Users.TryGetValue(request.Username, out var storedPassword)
        || storedPassword != request.Password)
        return null;
    // role from the caller-controlled username:
    new Claim(ClaimTypes.Role, request.Username == "admin" ? "Admin" : "User");
```

Why it matters.
Hardcoded plaintext credentials in source; non-constant-time != compare;
role derived from the username. Signing key also committed in appsettings.json.

2 · WEBAPIAUTHFREECLAUDE

Start clean, then ask for security

> "Generate a basic weather service as a C# web API."

> "Add token-based authorization to the web api."

What came back (Sonnet 4.6)

- **A clean JWT auth flow** /api/auth/login + protected /api/weather/{city}.
- **Functional…** compiles, runs, validates tokens.

…still shipped insecure

- **Hardcoded user dictionary** plaintext compare, no hashing.
- **Key committed; role from username** same default-insecure pattern as the baseline.

So I asked Claude to grade its own code: "analyse this code."

CLAUDE PRO · CLAUDE SONNET 4.6

SONNET 4.6

Generated project structure

PROMPT

"Generate a basic weather service as a C# web API" · "Add token-based authorization to the web API"

```
webapiauthfreeclaude/
├── Properties/launchSettings.json   — Dev launch config (ports 55978/55979)
├── Services/
│   ├── IWeatherService.cs           — Weather service interface + response model
│   └── MockWeatherService.cs        — Hardcoded stub implementation
├── AuthController.cs                — POST /api/auth/login endpoint
├── AuthModels.cs                    — LoginRequest / LoginResponse records
├── AuthService.cs                   — JWT token generation logic
├── IAuthService.cs                  — Auth service interface
├── Program.cs                       — DI wiring, middleware, JWT config
├── WeatherController.cs             — GET /api/weather/{city} (protected)
├── appsettings.json                 — JWT secret, issuer, audience settings
└── WeatherApi.csproj                — .NET 8.0, nullable enabled
```

CLAUDE PRO · SONNET 4.6

MIXED

Token validation — default configuration

```
var jwtSettings = builder.Configuration.GetSection("Jwt");
var key = Encoding.UTF8.GetBytes(jwtSettings["Key"]!);   // secret from appsettings.json

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwtSettings["Issuer"],
            ValidAudience = jwtSettings["Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(key)
        };
    });
```

- All four standard checks on (issuer, audience, lifetime, signing key)
- but the signing key is read from a secret committed in appsettings.json.
- Missing: RequireSignedTokens = true

CLAUDE PRO · SONNET 4.6

PROBLEMATIC

Authentication & token creation

```
private static readonly Dictionary<string, string> Users = new() {
    { "admin", "password123" },
    { "user",  "weather456"  }
};

public LoginResponse? Authenticate(LoginRequest request) {
    if (!Users.TryGetValue(request.Username, out var stored)
        || stored != request.Password)            // plain-text comparison
        return null;
    // ...
    var claims = new[] {
        new Claim(ClaimTypes.Name, request.Username),
        new Claim(ClaimTypes.Role, request.Username == "admin" ? "Admin" : "User"),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };
    var token = new JwtSecurityToken(issuer, audience, claims, expires, creds);
}
```

Hardcoded users with plain-text passwords (admin / password123),
compared directly with no hashing
Adding a unique GUID as Jti

REFLECTION · SONNET 4.6

SONNET 4.6

Asking the author to audit itself

PROMPT

“Claude, analyse this code.”

The same model that wrote the code is now asked to review it, in a new session.

CLAUDE PRO · SONNET 4.6

SONNET 4.6

Its own verdict: insecure by default

> **“Sonnet generated working but insecure code — optimizing for ‘runs immediately’ over ‘production-safe’.”**

What it detected

- Hardcoded credentials instead of environment config
- Plain-text password comparison, no hashing (BCrypt etc.)
- JWT secret placed directly in appsettings.json

What it highlights

- AI code isn’t self-correcting at generation time — the model rarely flags its own flaw unless asked
- Demo / tutorial framing is risky — “show me how JWT works” is treated as licence to skip security boilerplate

OPUS 4.7 · WEBAPIAUTHFREECLAUDE

“analyse this code” — with Opus 4.7

3

CRITICAL

- **Hardcoded creds** admin/password123 in source, no hashing
- **Signing key committed** “YourSuperSecretKey…” in appsettings.json
- **Role from username** admin string ⇒ Admin role

4

HIGH

- **No rate-limit / lockout** login open to brute force
- **User enumeration** timing differs on user-miss
- **No HSTS / headers / handler** UseHttpsRedirection only
- **No ClockSkew=0** 5-min default grace

5

MEDIUM

- **Null-forgiving config → NRE** no startup validation
- **No token revocation** 60-min JWT, no deny-list
- **No CORS policy** unused Admin role; log exposure

…

Top fixes, ordered (Claude): rotate key → user-secrets, fail-fast · remove hardcoded users → hashed store, role from record · add rate limiting + lockout · add HSTS + headers + ClockSkew=0 · strongly-typed JwtOptions validated at startup.

HSTS response header instructs the browser to remember this rule for a specified timeframe (HTTP Strict Transport Security)

My reply: “implement all.”

FIX ROUND 1 · “IMPLEMENT ALL”

Hardened in one pass — no new packages

Rate limiter + lockout

- **AddRateLimiter** IP-partitioned “login” policy, 10 req/min
- **Per-account lockout** 5 failures → 15-min lock
- **429 + Retry-After** via ProblemDetails

PBKDF2 password hashing

- **PBKDF2-SHA256** 100,000 iterations, 16-byte salt
- **Constant-time verify** CryptographicOperations.FixedTimeEquals
- **Timing-safe** dummy-hash on user-miss

…and the rest of the five fix groups

- **Fail-fast JwtOptions** ValidateOnStart blocks the old default key; key emptied from config
- **Security headers + HSTS** nosniff, X-Frame-Options, Referrer-Policy, COOP/CORP, CSP; ClockSkew=0
- **Hashed user store** role from the user record, not the username;

CLAUDE PRO

HARDENED

Credentials now live in configuration

```
// appsettings.Development.json   (dev only — appsettings.json ships these empty)
"Jwt": {
    "Key": "dev-only-secret-please-change-in-prod-32+chars",
    "Issuer": "WeatherApi", "Audience": "WeatherApiUsers", "ExpiryMinutes": 60
},
"Auth": {
    "Users": [
        { "Username": "admin", "Password": "password123", "Role": "Admin" },
        { "Username": "user",  "Password": "weather456",  "Role": "User"  }
    ]
}
```

> Dev users (with their Role) and the signing key moved into appsettings.Development.json;
> appsettings.json keeps Auth:Users and Jwt:Key empty.

CLAUDE PRO

HARDENED

Hashed at load · role from the store

InMemoryUserStore — seeds & hashes the configured users (Stored in plain in config file … seeded and hashed when loaded into memory…)

```
foreach (var seed in options.Value.Users)          // from "Auth:Users"
{
    _users[seed.Username] = new User(
        seed.Username,
        PasswordHashing.Hash(seed.Password),          // PBKDF2 · 100k iterations
        seed.Role);                                   // role kept on the record
}
```

AuthService — verifies, then puts the stored role in the claim

```
var user = _users.FindByUsername(username);
var ok = PasswordHashing.Verify(password, user?.PasswordHash ?? DummyHash);
// ... on success:
new Claim(ClaimTypes.Role, user.Role)                // role from the store, not inline
```

webapiauthfreeclaude · Claude Code

HARDENED

PHASE 2 · CLAUDE CODE CLI Hardening ladder · 2 of 4 · hardened baseline · Sonnet 4.6 + Opus 4.7

Program.cs

```
builder.Services
    .AddOptions<JwtOptions>()
    .Bind(builder.Configuration.GetSection(JwtOptions.SectionName))
    .ValidateDataAnnotations()
    .Validate(o => o.Key != "YourSuperSecretKeyThatIsAtLeast32CharsLong!",
        "Replace the default Jwt:Key with a secret from user-secrets, env vars, or a secret store.")
    .ValidateOnStart();
```

Why it matters.
Boot-time validation refuses to start on a missing, too-short, or default signing key — the exact key vibeasis shipped.

webapiauthfreeclaude · Claude Code

HARDENED

PHASE 2 · CLAUDE CODE CLI Hardening ladder · rung 2 of 4 · hardened baseline · Sonnet 4.6 + Opus 4.7

Security/PasswordHashing.cs

```
private const int Iterations = 100_000;
private static readonly HashAlgorithmName Algorithm = HashAlgorithmName.SHA256;
// ...
var salt = RandomNumberGenerator.GetBytes(SaltSize);
var hash = Rfc2898DeriveBytes.Pbkdf2(password, salt, Iterations, Algorithm, HashSize);
return $"{Iterations}.{Convert.ToBase64String(salt)}.{Convert.ToBase64String(hash)}";
// ...verify (constant-time):
var actual = Rfc2898DeriveBytes.Pbkdf2(password, salt, iterations, Algorithm, expected.Length);
return CryptographicOperations.FixedTimeEquals(actual, expected);
```

Why it matters.
PBKDF2-SHA256 + random salt + constant-time FixedTimeEquals.
Iterations 100k here; raised to 600k in the pro projects later.

CLAUDE PRO · Auto-generated TESTING

SONNET 4.6

Smoke tests it wrote on its own

Unprompted, it added an xUnit suite using **WebApplicationFactory<Program>** — real end-to-end requests against the running app.
Later inherited by the proPassword and proIAMaas builds, but never extended.

- ✓ `Login_WithValidCredentials_ReturnsToken` **200** valid admin login returns a non-empty token with a future expiry
- ✓ `Login_WithBadPassword_Returns401` **401** a wrong password is rejected
- ✓ `Weather_WithoutToken_Returns401` **401** the protected weather endpoint refuses anonymous calls
- ✓ `Weather_WithValidToken_Returns200` **200** with a bearer token it returns the requested city
- ✓ `SecurityHeaders_ArePresentOnEveryResponse` **headers** nosniff, X-Frame DENY, CSP, COOP/CORP, Referrer- & Permissions-Policy
- ✓ `Login_AfterFiveFailures_LocksAccountWith429AndRetryAfter` **429** five bad logins lock the account and return a Retry-After

BETWEEN THE REVIEWS · STILL IN THE FIX-ROUND-1 SESSION

A clean bump — packages, then .NET 10

> "bump"

> "bump the whole thing to .net 10 ?"

"bump" — dependency hygiene

- **JWT 7.0.3 → 7.6.0** clears advisory GHSA-59j7-ghrg-fj52; stayed in the 7.x line to match JwtBearer 8.0's family

"bump to .NET 10" — retarget

- **Checked SDK + runtime first** both present, then bumped
- **Framework + tied packages retargeted** net8.0 → net10.0
- **No source changes needed** JWT / auth / options / middleware compiled unchanged

Result: both builds clean — 0 warnings, 0 errors. Two short prompts, two commands, zero breakage.

REVIEW 2 · FRESH “CYBERSECURITY FIRM” PASS

A second, harsher look

2

CRITICAL

- **Secrets committed to git** dev key + creds tracked — permanent in history
- **No token revocation** logout is client-only; Jti never checked

5

HIGH

- **AllowedHosts: "*"** host-header injection
- **Lockout lost on restart** in-memory singleton
- **Rate limiter shared bucket** “unknown” IP fallback
- **…**

5

MEDIUM (+7 low)

- **PBKDF2 100k < NIST 600k**
- **CSP style-src unsafe-inline**
- **No maxlength → DoS**
- **parseJwt no error handling**
- **60-min token, no refresh**

It even graded round 1's own fixes: the new rate limiter shares an “unknown” bucket (H3) and PBKDF2 is still 100k vs NIST 600k (M1). Verdict: crypto & defense-in-depth are solid — the critical gaps are operational (secrets in git, no revocation, host-header trust).

FIX ROUND 2 · “WHICH OF THESE CAN YOU FIX?”

Triaging by what's actually fixable

> “Which of these problems can you fix?” → “Here's an honest breakdown.”

Can fix now (code / config)

- **C1** strip secrets, add to .gitignore
- **C2** jti deny-list checked in OnTokenValidated + /logout
- **H1 / H3 / H4 / H5** AllowedHosts · forwarded-headers + deny bucket · weather rate-limit · generic 404
- **M1** raise PBKDF2 → 600,000
- **M2–M5, L1–L6** CSP, maxlength, parseJwt, 15-min token, hygiene

Needs more than a code edit

- **Git history** secrets already committed
- **Secret rotation** operational, outside the app
- **Durable lockout (H2)** needs a shared store (Redis/SQL) — the proPassword step

The honest move: Claude separated code-level fixes it can apply from the operational ones it can't — so “done” didn't mean “safe” by default.

Interesting suggestions …

Security/PasswordHashing.cs — salted PBKDF2-SHA256, 600k iterations, constant-time verify:

```
var salt = RandomNumberGenerator.GetBytes(16);
var hash = Rfc2898DeriveBytes.Pbkdf2(password, salt, 600_000, SHA256, 32);
// verify:
return CryptographicOperations.FixedTimeEquals(actual, expected);
```

Program.cs — fail-fast configuration refuses to boot with the default key:

```
.AddOptions<JwtOptions>().Bind(...).ValidateDataAnnotations()
 .Validate(o => o.Key != "YourSuperSecretKey…",
           "Replace the default Jwt:Key with a real secret.")
 .ValidateOnStart();   // + per-user lockout, IP rate limiting, jti deny-list, CSP/HSTS
```


RUNG 3 · WEBAPIAUTHCLAUDEPROPASSWORD

Persistence & real revocation

> “Add persistence via sqlite.”

11m 13s
autonomous build

What came back

- **EF Core / SQLite user store** parameterized LINQ, unique NOCASE username index — no SQL-injection surface.
- **Token revocation** a jti deny-list + a logout endpoint, checked on every request.
- **Plus** a wwwroot front-end and smoke tests. (Self-reported one EF/SQLite DateTimeOffset gotcha and fixed it.)

Each rung is a strict superset of the last — same contracts, more defense-in-depth.

webapiauthclaudeproPassword · Claude Code

HARDENED

PHASE 2 · CLAUDE CODE CLI Hardening ladder · 3 of 4 · + SQLite & token revocation

Program.cs

```
OnTokenValidated = async ctx =>
{
    var denyList = ctx.HttpContext.RequestServices.GetRequiredService<ITokenDenyList>();
    var jti = ctx.Principal?.FindFirstValue(JwtRegisteredClaimNames.Jti);
    if (jti is not null && await denyList.IsRevokedAsync(jti))
        ctx.Fail("Token has been revoked.");
}
```

Why it matters.
Server-side revocation: every request checks the jti deny-list (SQLite); logout revokes until the token's exp.

WEBAPIAUTHCLAUDEPROIAMAAS

Federated identity (Auth0)

› "I want to add an option to login via auth0."

Claude paused to scope it

- **"A couple of design questions first…"**
- **Tenant?** → already have one
- **Coexist with local login?** → both, side-by-side
- **Identity?** → trust Auth0 claims directly

What came back

- **Auth0 as a 2nd issuer** RS256 validated via JWKS — no new NuGet.
- **Policy-scheme selector** routes each token to local HS256 or Auth0.
- **SPA SDK front-end + logout** deny-list honoured for both schemes.

Federated IAM-as-a-service, sitting alongside the In-App hardened auth.

CODEGUARD · AUTH0 INTEGRATION LOG

AUTH0 · CSP

Auth0's CDN script vs a strict CSP

Adding an Auth0 login button — its SPA SDK loaded from **cdn.auth0.com** — collided with the API's locked-down Content-Security-Policy. Three conflicts, in order:

WHAT BROKE

1. default-src 'self' blocked the SDK download `Refused to load auth0-spa-js from cdn.auth0.com`
2. Wildcard matched only one subdomain level `*.auth0.com did NOT cover dev-xxxx.us.auth0.com (two levels deep)`
3. The SDK's hidden silent-auth iframe was blocked `"Framing 'https://…auth0.com/' violates CSP default-src 'self'" — frame-src falls back to default-src`

THE CSP THAT FIXED IT

```
default-src 'self';
script-src  'self' https://cdn.auth0.com;          // load the SPA SDK  (+ pin an SRI hash)
frame-src   https://dev-xxxx.us.auth0.com;         // hidden silent-auth iframe
connect-src 'self' https://dev-xxxx.us.auth0.com;  // /authorize + token calls
```


webapiauthclaudeproIAMaas · Claude Code

MIXED

PHASE 2 · CLAUDE CODE CLI Hardening ladder · rung 4 of 4 · + Auth0 federation

Program.cs

```
options.ForwardDefaultSelector = context =>{
    var auth = context.Request.Headers.Authorization.FirstOrDefault();
    if (auth?.StartsWith("Bearer ", StringComparison.OrdinalIgnoreCase) == true
        && auth0Config.IsConfigured){
        var token = auth["Bearer ".Length..].Trim();
        var handler = new JwtSecurityTokenHandler();
        if (handler.CanReadToken(token))
        {
            var jwt = handler.ReadJwtToken(token);    // no signature check here
            if (jwt.Issuer.Equals($"https://{auth0Config.Domain}/", StringComparison.OrdinalIgnoreCase))
                return "Auth0";
        }
    }
    return JwtBearerDefaults.AuthenticationScheme;
};
```

Why it matters.

Federated IAM: routes each token to the local HS256 or Auth0 RS256 scheme by issuer.
Elegant, but it reads the issuer from an unvalidated token (each scheme still verifies its own signature).

COMPARISON

Security capability matrix

Control	G1	G2	G3	G4	G5	C1	C2	C3	C4
Secret kept out of repo	✗	✗	✗	✗	✓	✗	✓	✓	✓
Password hashing (PBKDF2)	✗	✗	✗	✗	✗	✗	✓	✓	✓
Brute-force lockout	✗	✗	✗	✗	✗	✗	✓	✓	✓
Rate limiting	✗	✗	✗	✗	✗	✗	✓	✓	✓
Token revocation / logout	✗	✗	✗	✗	✗	✗	✗	✓	✓
Security headers + HSTS	✗	✗	✗	✗	✗	✗	✓	✓	✓
Startup key validation	✓	✗	✗	~	✓	✗	✓	✓	✓
Real credential validation	~	~	✗	~	~	~	✓	✓	✓
Audience validation	✓	✓	✗	✓	✓	✓	✓	✓	✓

CONCLUSION & HACKATHON

A “junior-dev with an agent” workflow

Terse, high-trust prompts that lean on the agent to enumerate, implement, and critique its own work — with dev as the test gate and the occasional supplier of a structured spec. **It produced a genuinely hardened lineage.**

Its blind spots — secrets in source, supply-chain pinning, in-memory security state — were caught by **review rather than prevented by design**: precisely the gap the planned spec-driven, harness-based next iteration is meant to close.

HACKATHON · THE BRIEF

Your mission

Add token-based authorization to one of your own web APIs — entirely through prompts — and drive it all the way to a **secure baseline**, not just “it runs”.

Solo

One developer, one project — your call on language & framework.

Your own API

Bring a real web service you already have (or starting).

Any AI tool

Copilot, Claude Code, Cursor… whatever you use.

2 hours

Generate → audit → harden → verify.

Everything goes through prompts. Keep your prompt log — it is half the lesson.

65

HACKATHON · BEFORE YOU START

Set up & share your repo

Three things to do up front — the history and the prompts are part of the experiment.

1

Push to GitHub / GitLab

Create a repo for your project before you write a line — public or private, your choice.

2

Commit frequently

Small, frequent commits. The commit history is what we learn from — don't squash it.

3

Keep a prompt log

Save the prompts that worked in a PROMPTS.md — the journey matters as much as the code.

Please share your repo URL with us (bert.lagaisse@kuleuven.be).
It will be treated as confidential and used only for anonymous analytics on commit patterns and security architecture.

66

HACKATHON · HOW IT RUNS

Two hours, four beats

Setup	Pick the project, get your AI tool ready, start a prompt log.	10 min
1 · Generate	One prompt: add token-based auth. Get login → JWT → a protected endpoint working.	20 min
2 · Audit	Ask the same model to review its own code against the OWASP Top 10. Capture findings. Refine the prompt if needed.	20 min
3 · Harden	Fix the findings one at a time: secret out of source, hash passwords, real validation, claims.	45 min
4 · Verify	Run a misuse-case test and a secret scan; check against the definition of done.	15 min
Wrap	Quick demo; compare how your tool behaved with your neighbour's.	10 min

HACKATHON · STARTER PROMPTS

Prompts to get you moving

ROUND 1 · GENERATE

"Add token-based authorization to this web API: a /login endpoint that issues a JWT, and protect the existing endpoints with the framework's standard JWT middleware."

ROUND 2 · AUDIT

"You are a senior security engineer. Review the authentication code you just wrote against the OWASP Top 10. List each concrete vulnerability and the fix — don't change anything yet."

ROUND 3 · HARDEN (repeat per finding)

"Fix issue #1 only: move the signing key out of source into configuration / user-secrets, require ≥ 256-bit, and fail startup if it's missing. Then show me the diff."