

Shared Version



Dark Side of AI

Writing Insecure Applications In Minutes



Chris Lindsey
Field CTO – OX Security

Thank You for attending, “The Dark Side of AI: Writing Insecure Applications in Minutes”

A quick message from me (Chris):

I really enjoy sharing with the security community. It's a passion that I have and part of my mission to help make applications more secure. I put a lot of effort in writing this talk, this sharable version is only for those who personally attended my talk. Please do not share or use these for your own talk without reaching out to me first.

Note about this sharable version of the slides:

This presentation does not include the screen shots or the videos that were shown during the presentation. This was done to make the deck easier to share. (Actually, made it extremely large). The core pieces of this talk are included. You can take the prompts and try them out for yourself. The results should be very similar.

I am always open to giving this talk: to a community, company or group of folks who want to hear it. Feel free to reach out to me.


I can be reached either at chris.lindsey@ox.security or via LinkedIn <https://www.linkedin.com/in/chris-lindsey-39b3915>.

 **OX**security



Agenda

- Introduction
- Common Prompts
- Writing Software using AI
- Reviewing Generated Code
- Going Deeper with Security
- Final Thoughts
- Q & A



IF YOU CHOOSE
NOT TO DECIDE,
YOU STILL HAVE
MADE A CHOICE

- NEIL PEART

Introduction

Why this matters

 oxsecurity

Introduction

AI tools have become integral to modern software development, enhancing productivity and code quality. Recent surveys and studies provide insights into the extent of AI adoption among developers:

- **76%** of developers will use development AI tools this year, **70%** last year.
 - [Stack Overflow Developer Survey 2024](#)
- **97%** of developers have used AI coding tools at some point.
- **40%** employers actively encourage and promote AI tool adoption.
 - [GitHub Survey](#)



Introduction

AI tools have become integral to modern software development, enhancing productivity and code quality. Recent surveys and studies provide insights into the extent of AI adoption among developers:

- Google -> AI Generates **25%** of new code
- Gartner -> By 2027, **50%** of code engineering will use AI
 - Up from **5%** in 2024



These findings underscore the growing integration of AI tools in software development, highlighting their role in enhancing efficiency and code quality.

Common Prompts

Quick Prompting 101



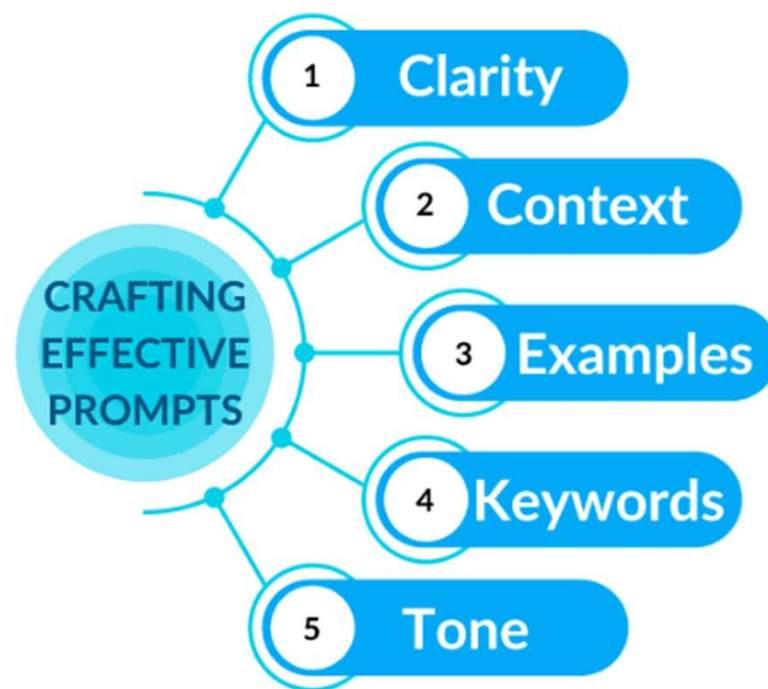
Common Prompts

Prompting is an art and science

How you prompt is just as important as how you code. Un-Focused prompting will result in sloppy results, while strategic prompts will result in clean and focused results.

Tips for better prompting:

- Be as **specific** as possible
- Supply AI with **examples** when possible
 - Example methods, formats that you're looking for



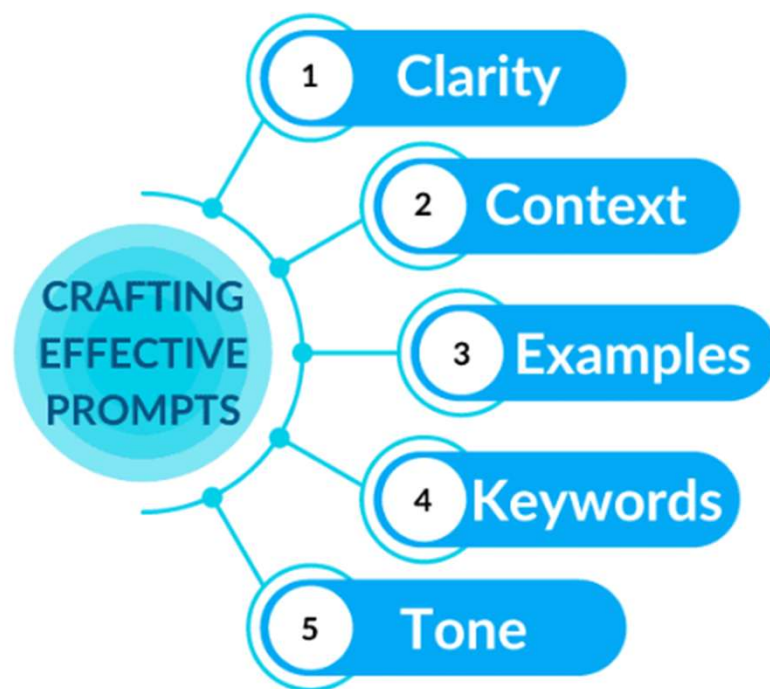
Common Prompts

Prompting is an art and science

How you prompt is just as important as how you code. Un-Focused prompting will result in sloppy results, while strategic prompts will result in clean and focused results.

Tips for better prompting:

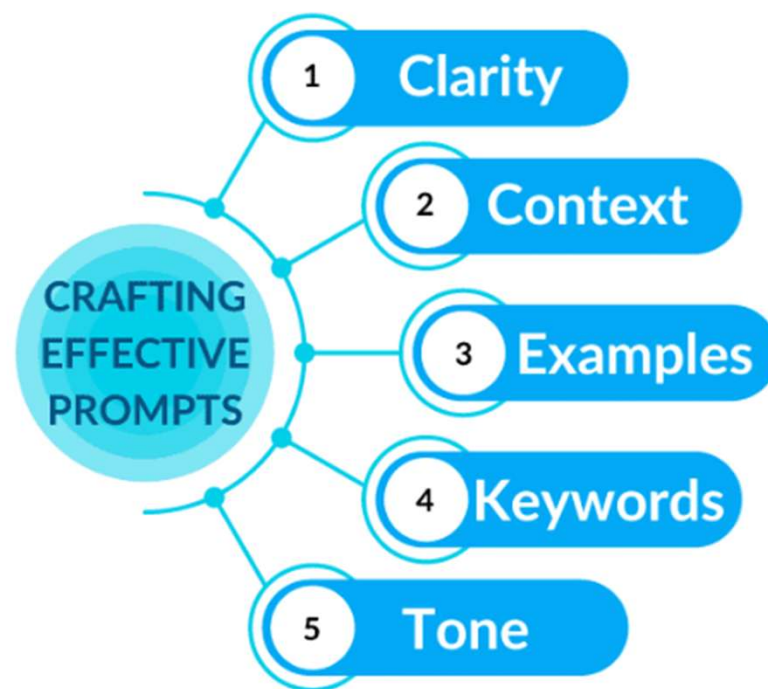
- Provide **data**
 - Tables, structures, details
- Provide “**what to do**” instead of what “Not to do”



Common Prompts

Prompting is an art and science

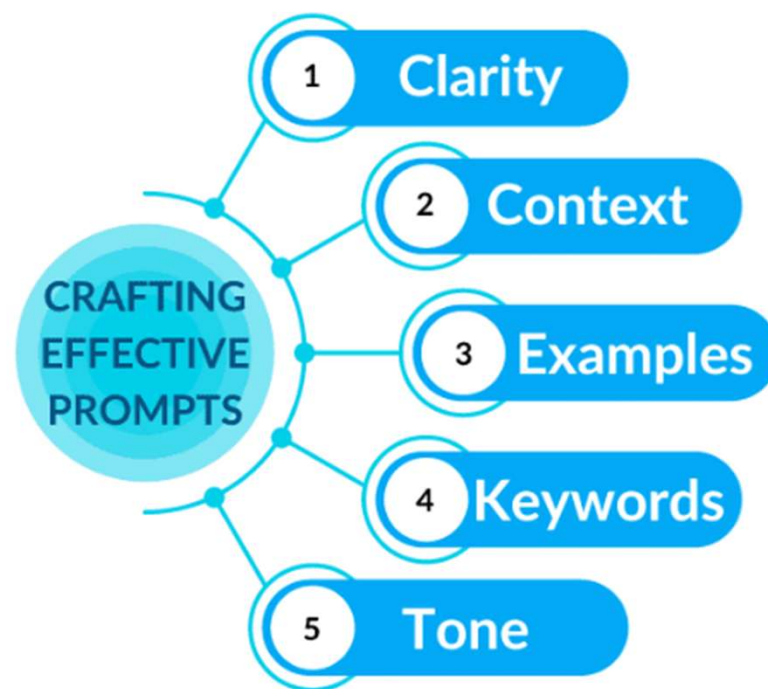
- Try **chain of thought** prompting
 - It's ok to spoon feed the prompt, you don't have to give it a paragraph to work from
- **Split complex tasks** into simpler ones



Common Prompts

Prompting is an art and science

- Understand that the **results might be wrong** and you have to tell the model
 - Asking the model, “Are you sure” or “Is this right?”



Common Prompts

Basic Prompting = Basic Results

Common Prompts

These examples are commonly used and are too vague. While these can be useful, they lack common key elements that would make better results.


Basic API Setup

1. "Generate a basic C# .NET Core Web API project with CRUD operations for a **Person** entity using Entity Framework Core and SQL Server."
2. "Create a C# Web API in .NET Core that connects to a SQL Server database and performs CRUD operations using stored procedures."
3. "Generate a C# .NET 6 Web API template with proper project structure and dependency injection."

Controllers and Routing

4. "Write a C# Web API controller for managing **Person** entities with endpoints for Create, Read, Update, and Delete."
5. "Generate a C# API controller with attribute-based routing and versioning in .NET Core."

Data Access Layer (DAL)

6. "Create a repository pattern in C# for handling database operations in a .NET Core Web API project."
7. "Generate a C# data abstraction layer for a .NET Core Web API using stored procedures." 
8. "Write a C# service layer that interacts with the data layer and provides business logic for an API."

Common Prompts

These examples are commonly used and are too vague. While these can be useful, they lack common key elements that would make better results.

Database Integration

9. "Create a C# Entity Framework Core DbContext and models for a **Person** table with name, address, city, state, zip code, and phone number."
10. "Generate SQL Server stored procedures for CRUD operations and a corresponding C# data access layer to interact with them."

Security & Authentication

11. "Implement JWT authentication in a C# .NET Core Web API with login and registration endpoints."
12. "Secure a C# Web API using API keys and middleware for request validation."
13. "Implement role-based authorization in a .NET Core Web API using JWT and ASP.NET Identity."

Common Prompts

These examples are commonly used and are too vague. While these can be useful, they lack common key elements that would make better results.

Validation & Error Handling

- 14. "Add FluentValidation to a C# .NET Core Web API for validating request models."
- 15. "Implement global exception handling in a .NET Core Web API using middleware."

Testing & Documentation

- 16. "Generate Swagger/OpenAPI documentation for a C# .NET Core Web API."
- 17. "Write unit tests for a C# Web API controller using xUnit and Moq."

Common Prompts

Better Prompting = Less
hours spent fixing AI
code

Common Prompts – Better Focus

These examples are more specific. These can be used as starter prompts to generate better results than previous examples.

Frontend Framework Selection & Setup

1. "Generate a React frontend that connects to a .NET Core API for CRUD operations on a **Person** entity."
2. "Set up an Angular application that fetches and updates data from a C# Web API."
3. "Create a simple Vue.js application that interacts with a .NET Core API to display and modify person data."

Fetching Data from API

4. "Write a React component that fetches data from a .NET Core Web API using Axios and displays it in a table."
5. "Generate an Angular service to interact with a .NET Core Web API for handling **Person** CRUD operations."
6. "Create a JavaScript fetch request to get data from a .NET Core Web API and display it on a webpage."

Form Handling & Data Submission

7. "Create an Angular reactive form to submit data to a C# Web API and display success/error messages."
8. "Generate a React form with validation to add a new **Person** record via a .NET Core API."
9. "Write a Vue.js component that binds a form to an API call for adding a new record."

Common Prompts – Better Focus

These examples are more specific. These can be used as starter prompts to generate better results than previous examples.


State Management & UI Libraries

10. "Use React Query to manage API data fetching and caching in a React frontend that connects to a .NET Core backend."
11. "Generate a Redux store and actions for managing state from a .NET Core API in a React app."
12. "Set up NgRx in an Angular app to manage API responses from a .NET Core backend."

Authentication & Security

13. "Implement JWT authentication in a React app that consumes a .NET Core API."
14. "Create an Angular authentication service that handles login/logout with a JWT-secured C# Web API."
15. "Secure API calls in a React app using OAuth 2.0 authentication with a .NET Core backend."

Error Handling & User Feedback

16. "Write a React error boundary component to handle API request failures and display user-friendly messages."
17. "Generate an Angular interceptor for handling HTTP errors and displaying toast notifications." 
18. "Implement a global error handling mechanism in a Vue.js app that interacts with a .NET Core API."

Common Prompts – Better Focus

These examples are more specific. These can be used as starter prompts to generate better results than previous examples.

Tables, Pagination & Filtering

19. "Create a React DataTable component that fetches paginated data from a .NET Core API."
20. "Generate an Angular Material table with sorting, filtering, and pagination that connects to a .NET Core API."
21. "Write a Vue.js component with a search bar and filters to query a .NET Core backend."

Deployment & Optimization

22. "Build and deploy a React frontend and .NET Core backend as a full-stack web application using Docker."
23. "Optimize API calls in a React app using caching and memoization techniques."
24. "Set up CI/CD for an Angular frontend and .NET Core backend using GitHub Actions."

Writing Software using AI

How you prompt is the
difference between
success and failure

 oxsecurity

Writing Software using AI

Back End – ChatGPT

 oxsecurity

Writing Software Using AI - ChatGPT

ChatGPT appears to be drunk, my prompt: Update the webpage to include the fields in the person model and add a button that calls that api back end.

```
public async Task<IActionResult> OnPostGetPeople()
{
    if (string.IsNullOrEmpty(Token))
    {
        return RedirectToPage("/Index");
    }

    var httpClient = _httpClientFactory.CreateClient();
    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);
    var apiUrl = _configuration["ApiUrl"];

    var response = await httpClient.GetAsync($"{apiUrl}/api/person");

    if (response.IsSuccessStatusCode)

public async Task<IActionResult> OnPostAddPerson()
{
    if (string.IsNullOrEmpty(Token))
    {
        return RedirectToPage("/Index");
    }

    var httpClient = _httpClientFactory.CreateClient();
    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);
    var apiUrl = _configuration["ApiUrl"];
    var json = JsonSerializer.Serialize(NewPerson);
    var content = new StringContent(json, Encoding.UTF8, "application/json");
    var response = await httpClient.PostAsync($"{apiUrl}/api/person", content);

    if (!response.IsSuccessStatusCode)
    {
```

OnPostGetPeople()
was removed and
replaced with
OnPostAddPerson()

Most of the method
has the previous
code – what
happened here?

Writing Software using AI

Back End – Using Claude 3.7

 oxsecurity

Writing Software Using AI – Back End (Claude 3.7)

Goal: To mimic how a normal developer attempts to write a basic application that contains both a front end and API back end.

The following prompts will be used to build today's application. Between each prompt, we will discuss what we see.

1. **Generate a basic C# .NET Core Web API project with CRUD operations for a Person entity using Entity Framework Core and SQL Server.**
 - a. What's missing?
 - i. Role Based Access, Security Tokens, Data validation, Unit Test Cases, lack of error handling, database security (Stored Procedures, etc...)
2. **What security issues do you see with this? What is missing and what can I do to make it more secure?**
 - a. Getting closer, but still missing key security pieces such as Tokens, RBA and more listed from above
3. **Please secure the code from the identified security findings. Also, implement JWT Tokens including endpoints to log on and retrieve a JWT token.**
4. **What security issues do you see with this? What is missing and what can I do to make it more secure?**
5. **Please secure the code from the identified security findings.**
6. **Can you provide the application files in download form for me to open this application in visual studio?**
 - a. Result of this was the steps to install

Writing Software using AI

Front End – Using Claude 3.7

 oxsecurity

Writing Software Using AI – Front End (Claude 3.7)

The following prompts will be used to build today's application. Between each prompt, we will discuss what we see.

1. **Generate a front end web based application that ties into this backend api that you generated for me. This should be in the same language and have the hooks to connect into the api that you just provided to me.**
2. **What security issues do you see with this? What is missing and what can I do to make it more secure?**
 - a. Findings: Token Storage in public property, Missing CSRF Protection, Weak Credential Handling, No Input Validation, JWT Token Expiration, No rate limiting, Session Fixation Vulnerability
3. **Please secure the code from the identified security findings**
 - a. JsonSerializer.Deserialize is not sanitized, so this could be an issue
 - b. Missing person validation – Front end should still attempt to validate the same thing the back end should

Building Generated Code

The following prompts will be used to build today's application:

1. Open Command Prompt as administrator and run these commands
 - a. `mkdir PersonApi`
 - b. `cd PersonApi`
 - c. `dotnet new sln`
 - d. `dotnet new razor -o PersonApiFrontend`
 - e. `dotnet sln add PersonApiFrontend`
 - f. `dotnet new razor -o PersonApiBackend`
 - g. `dotnet sln add PersonApiBackend`
2. Open Visual Studio and open new sln
3. Build Solution



Writing Software using AI

Summary: Reviewing Code

 oxsecurity

Reviewing Generated Code

Findings: Like transformers, more than meets the eye!

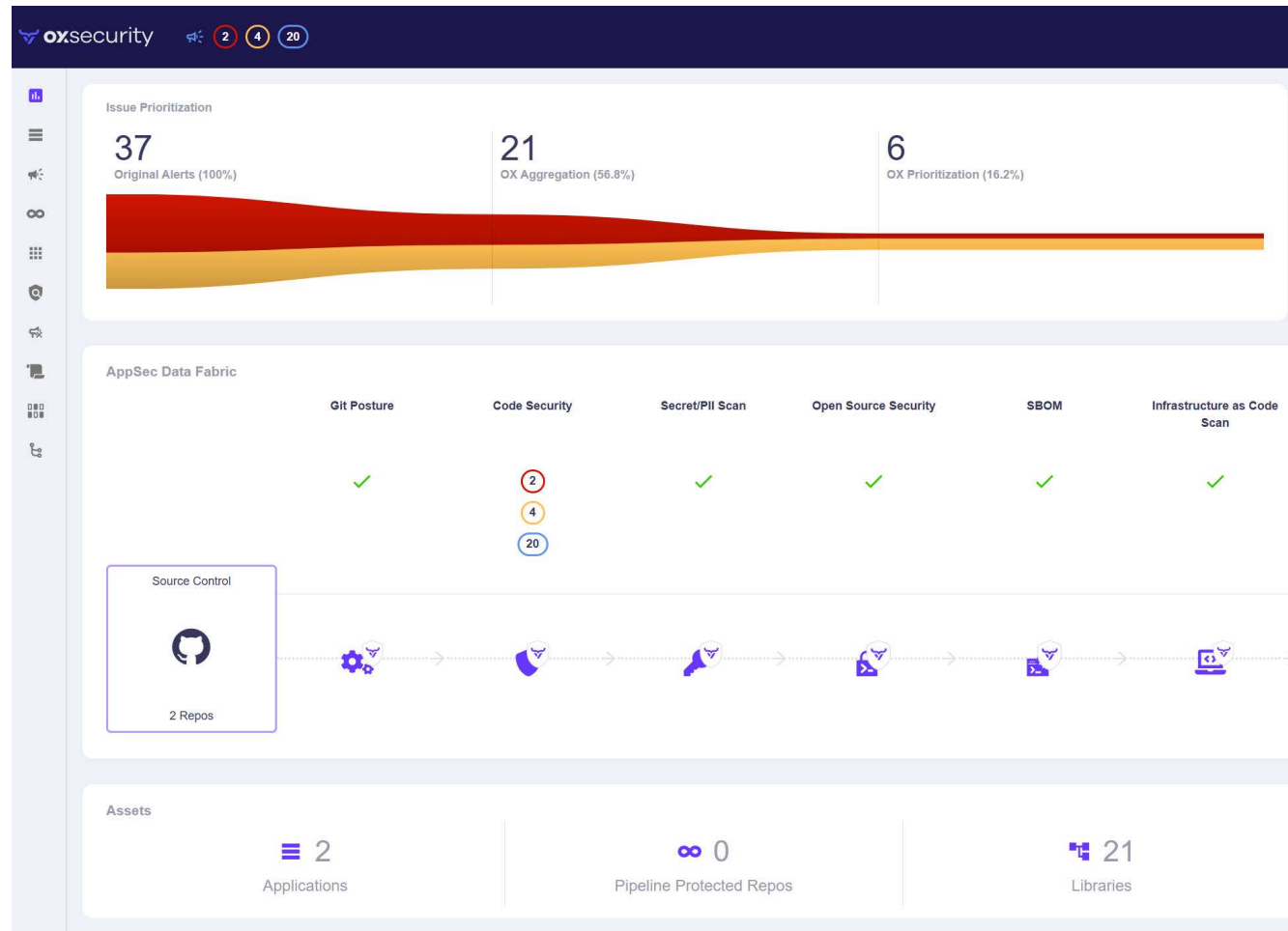
Where the rubber hits the road – Observations:

1. You can't expect any "GPT" system to write your application for you
 - a. **Every** "GPT" system
 - i. **created code** that looked good, in pieces
 - ii. Generated **insecure** code, even when told to write secure code
 - iii. Even after **multiple attempts** asking "GPT" if the code had security issues, it still kept coming up with more. It would fix some, but then create or revert back other fixes.
 - b. **ChatGPT** – Acted like a drunk coder, however, code did compile. It was missing the vast majority of what we asked for even when asked to create secure code.
 - c. **Claude** – Senior developer who has not been trained on security, tries it's best, but falls just a little short
 - d. **Copilot** – I'll give you code that looks right, but good luck trying to build what I gave you




Going Deeper with Security

**“GPT” can only
take you so far,
how about
enterprise security
tools?**




Going Deeper with Security

“GPT” can only take you so far, how about enterprise security tools?

#	Severity 	Category	Name
1	● High	</> Code Security	JWT token's expiry date is not validated
2	● High	</> Code Security	JWT token's expiry date is not validated
3	● Medium	</> Code Security	Mass Assignment Vulnerability in ASP.NET Core
4	● Medium	</> Code Security	Log injection vulnerability due to unvalidated user input
5	● Medium	</> Code Security	Mass Assignment Vulnerability in ASP.NET Core
6	● Medium	</> Code Security	Log injection vulnerability due to unvalidated user input
7	● Low	</> Code Security	Insecure JWT validation
8	● Low	</> Code Security	Path Traversal vulnerability due to unvalidated user input
9	● Low	</> Code Security	Cross Site Request Forgery (CSRF) vulnerability due to missing anti-forgery token
10	● Low	</> Code Security	Insecure Deserialization due to JsonConvert usage
11	● Low	</> Code Security	Cross Site Scripting (XSS) vulnerability due to interpolated variables
12	● Low	</> Code Security	Stack Trace Information Leakage in Non-Development Environment



Going Deeper with Security

“GPT” can only take you so far, how about enterprise security tools?

#	Severity 	Category	Name
13	● Low	</> Code Security	XSS vulnerability due to unvalidated user input
14	● Low	</> Code Security	Denial-of-Service (DoS) vulnerability due to regex input
15	● Low	</> Code Security	Regular Expression Injection
16	● Low	</> Code Security	Non-structured logging in C#
17	● Low	</> Code Security	Insecure JWT validation
18	● Low	</> Code Security	Path Traversal vulnerability due to unvalidated user input
19	● Low	</> Code Security	XSS vulnerability due to unvalidated user input
20	● Low	</> Code Security	Cross Site Request Forgery (CSRF) vulnerability due to missing anti-forgery token
21	● Low	</> Code Security	Insecure Deserialization due to JsonConvert usage
22	● Low	</> Code Security	Cross Site Scripting (XSS) vulnerability due to interpolated variables
23	● Low	</> Code Security	Stack Trace Information Leakage in Non-Development Environment
24	● Low	</> Code Security	Denial-of-Service (DoS) vulnerability due to regex input

Going Deeper with Security

“GPT” can only take you so far, how about enterprise security tools?

#	Severity 	Category	Name
25	● Low	</> Code Security	Regular Expression Injection
26	● Low	</> Code Security	Non-structured logging in C#
27	● Info	</> Code Security	Missing or Broken Authorization in .NET Applications
28	● Info	 Open Source Security	jose-jwt@5.1.1 is a C# direct dependency having 1 indirect vulnerability and no direct vulnerabilities. The dependency is not imported into the code.
29	● Info	</> Code Security	Missing or Broken Authorization in .NET Applications

Going Deeper with Security

What does ChatGPT have to say on this matter?



Going Deeper with Security

“GPT” can only take you so far, how about enterprise security tools?

Enterprise security tools, like SAST (Static Application Security Testing), DAST (Dynamic Application Security Testing), and SCA (Software Composition Analysis), are significantly better at finding security vulnerabilities than AI coding assistants like ChatGPT, Copilot, or Claude. Here's why:

1. Depth and Accuracy of Analysis

- Enterprise Security Tools
 - Designed specifically for security scanning, using pattern matching, taint analysis, control flow analysis, and machine learning models trained on security-specific datasets.
 - Can find vulnerabilities like SQL injection, XSS, hardcoded secrets, and insecure cryptographic implementations with high accuracy.
 - Provide compliance and regulatory checks (e.g., OWASP Top 10, NIST, PCI DSS).
- AI Coding Assistants
 - Trained on a mix of secure and insecure code, meaning they can generate both good and bad code.
 - Can highlight basic syntax errors and sometimes suggest secure coding practices, but they don't perform in-depth static analysis.
 - Lack deep codebase context, making them prone to missing vulnerabilities that require inter-file or dependency-level understanding.

Going Deeper with Security

Enterprise Tools vs “GPT”

2. Codebase Coverage

- Enterprise Security Tools
 - Can scan entire repositories, tracking vulnerabilities across multiple files, functions, and even microservices.
 - Can integrate with CI/CD pipelines to enforce security policies.
- AI Coding Assistants
 - Only analyze small snippets of code at a time.
 - Lack the ability to track vulnerabilities across multiple files or detect issues in a full codebase.

3. False Positives & False Negatives

- Enterprise Security Tools
 - While they can produce false positives, modern SAST tools are tunable and can filter out noise with rule-based configurations.
 - They also leverage data flow analysis to identify true vulnerabilities more effectively.
- AI Coding Assistants
 - Often suggest code that “looks correct” but may introduce subtle vulnerabilities.
 - Don’t perform proper taint analysis or data flow tracking, leading to missed issues (false negatives).

Going Deeper with Security

Enterprise Tools vs “GPT”

4. Security-Specific Knowledge

- Enterprise Security Tools
 - Continuously updated with vulnerability databases (e.g., CVEs, CWE, OWASP).
 - Can detect newly disclosed vulnerabilities in third-party libraries and dependencies.
- AI Coding Assistants
 - Not security-focused and may generate code that reintroduces known vulnerabilities.
 - Don't check against vulnerability databases in real-time.

5. Regulatory and Compliance Support

- Enterprise Security Tools
 - Help enforce compliance with industry security standards and best practices.
 - Can generate security reports for audits.
- AI Coding Assistants
 - Cannot enforce compliance or generate security reports.

Final Thoughts

- There is NO magic bullet when it comes to development and AI
- Human intervention is key for success
- Just because you can, does not mean you should

Thank You!



Chris Lindsey
Field CTO – OX Security



<https://www.linkedin.com/in/chris-lindsey-39b3915>