Security Best Practices for Indie Hackers Using Vibe Coding

Vibe coding—the practice of using AI to generate code based on conversational prompts—has revolutionized how indie hackers and non-technical founders build software. As of March 2025, this approach has gained significant traction, allowing builders with minimal coding experience to create functional applications. However, this accessibility comes with security considerations that shouldn't be overlooked. This report presents practical, beginner-friendly security practices to help you build safer applications while enjoying the benefits of AI-powered development.

Understanding the Security Risks in Vibe Coding

Vibe coding introduces unique security challenges that differ from traditional development. The "accept all" mentality, where developers trust AI suggestions wholesale without review, can lead to vulnerabilities that may go unnoticed[1].

"Vibe coding is all fun and games until you have to vibe debug," as noted by industry experts[2]. The challenge lies in the fact that AI might produce code that "looks right" and "mostly works" while harboring security flaws that non-technical builders might miss[2]. Security vulnerabilities can slip through without diligent review, especially when developers blindly accept AI suggestions[1].

Common Security Pitfalls

AI-generated code may contain several security issues:

- Outdated cryptography implementations
- Missing input validation
- Vulnerable authentication mechanisms
- Insecure API handling
- Improper error handling that exposes sensitive information
- Dependency vulnerabilities

As an indie hacker with limited technical knowledge, you might not immediately recognize these issues, making it crucial to adopt preventive practices.

Essential Security Practices for Beginners

1. Treat AI as a Junior Developer

Always approach AI-generated code with a critical eye. Even without deep technical knowledge, you can develop a healthy skepticism toward code solutions[2].

Practical Example:
When asking Claude or GPT to create a login system, don't just implement the first solution provided. Instead, follow up with:

```
"Before I implement this login system, can you highlight any potential security concerns
```

This simple follow-up prompt often reveals important security considerations the AI might have overlooked in its initial response.

2. Use Explicit Security-Focused Prompts

The quality of AI-generated code directly corresponds to the quality of your prompts. Being explicit about security requirements yields more secure code.

Practical Example:
Instead of:

```
"Create a contact form for my website"
```

Try:

```
"Create a contact form for my website that validates and sanitizes all user inputs, preve
```

This specificity guides the AI to implement security measures from the beginning.

3. Implement the Iterate-and-Verify Approach

While vibe coding encourages rapid iteration, each cycle should include security verification.

Practical Example:
After generating a feature, ask:

```
"Please review this code for security vulnerabilities, particularly around data handling
```

Then follow with:

```
"How would a malicious user potentially exploit this code? Please suggest fixes"
```

This two-step prompt pattern helps identify and address vulnerabilities that might not be obvious at first glance.

Practical Security Implementation for Common Features

Securing User Authentication

Authentication is a critical security component in most applications and is frequently implemented incorrectly in AI-generated code.

Practical Implementation:

1. Request a login system with explicit security requirements:

```
"Create a secure login system that uses bcrypt for password hashing, implements rate limi
```

2. Verify the implementation with this follow-up:

```
"Show me how this login system protects against common attacks like credential stuffing,
```

Securing Data Storage and Handling

Even with minimal technical knowledge, you can ensure proper data handling.

Practical Implementation:
When asking the AI to implement data storage:

```
"Create a function to store user profile data that encrypts sensitive information, saniti
```

Then verify:

```
"Review this data storage implementation and explain how it protects against data leakage
```

Implementing Secure API Connections

Many indie projects rely on third-party APIs, which can be a source of security vulnerabilities.

Practical Implementation:
When implementing API connections:

```
"Create a secure function to connect to the Stripe API that handles API keys safely, vali
```

Advanced Security Considerations for Growing Projects

As your project grows, additional security measures become important.

Implement Environment-Based Configuration

Practical Example:
Ask your AI assistant:

```
"Set up a secure environment configuration system that keeps API keys, database credentia
```

Regularly Update Dependencies

Outdated libraries often contain known vulnerabilities.

Practical Implementation:
Set a calendar reminder for monthly security audits. During each audit, ask:

```
"Please check all dependencies in my project for known security vulnerabilities and sugge
```

Set Up Basic Automated Security Testing

Even without technical knowledge, you can implement simple security testing.

Practical Implementation:
Ask your AI assistant:

```
"Create a basic GitHub Actions workflow that scans my code for security vulnerabilities u
```

Building a Security Mindset as a Non-Technical Founder

Developing security awareness is crucial for long-term project health.

Learn from AI Explanations

Use each interaction as a learning opportunity by asking for explanations.

Practical Example:
After implementing a security feature, ask:

```
"Explain why this security practice is important and what vulnerabilities it protects aga
```

Create a Security Checklist for Your Project

Develop a custom security checklist tailored to your specific application.

Practical Implementation:
Ask:

```
"Based on my project—a membership site with user accounts, payment processing, and conter
```

Document Security Decisions

Maintain documentation of security decisions for future reference and team onboarding.

Practical Example:
After implementing important security features, ask:

```
"Summarize the security measures we've implemented and the rationale behind each one in a
```

Conclusion

Vibe coding makes building software more accessible than ever before, but it doesn't eliminate the need for security consciousness. By implementing these beginner-friendly security practices, indie hackers can significantly reduce their vulnerability risk while still enjoying the rapid development benefits of AI-assisted coding.

Remember that security is an ongoing process, not a one-time implementation. As your project grows, revisit these practices and expand your security measures accordingly. With the right approach, you can leverage AI to build not just functional applications, but secure ones too.

The most important takeaway is to maintain a balance between the "vibe coding" ethos of quick iteration and the necessity of security vigilance. By incorporating these simple security checks into your workflow, you're not slowing down development—you're ensuring your product's long-term success and protecting your users' trust.

⁂

1. https://zbrain.ai/what-is-vibe-coding/
2. https://blankslatedigital.co.uk/blog/artificial-intelligence/what-is-vibe-coding/