

The background is a dark, textured surface with a faint grid pattern. A globe is visible in the lower half, showing the Americas. Several padlock icons are scattered across the upper half, some within hexagonal shapes. Binary code (0s and 1s) is faintly visible in the background.

Open Source Security and Generative AI: Navigating Concerns while Enhancing Development


with Javier Perez



Javier Perez

Perforce Software

 Chief Evangelist & Sr. Director of Product Management

 javierperez.mozello.com

 www.linkedin.com/in/javierperez

Tech Evangelist | Open Source Advocate | OSS Program Leader | Product Management
Product Marketing | App Security | Speaker | Blogger | Startups | Ex-Red Hat | Ex-IBM

npm

2.6M +

720 packages/day



Packagist

384K +

86 packages/day

Maven™

575K +

182 packages/day

nuget

379K +

161 packages/day

PyPI

475K +

211 packages/day

RubyGems

178K +

12 packages/day



350+

Projects



850+

Projects



418+

Projects



157+

Projects



30+

Projects



CD.FOUNDATION



Open Source Software



Source code is publicly available

Open to collaboration

Source code available with a license that permits users to freely run, modify, and redistribute

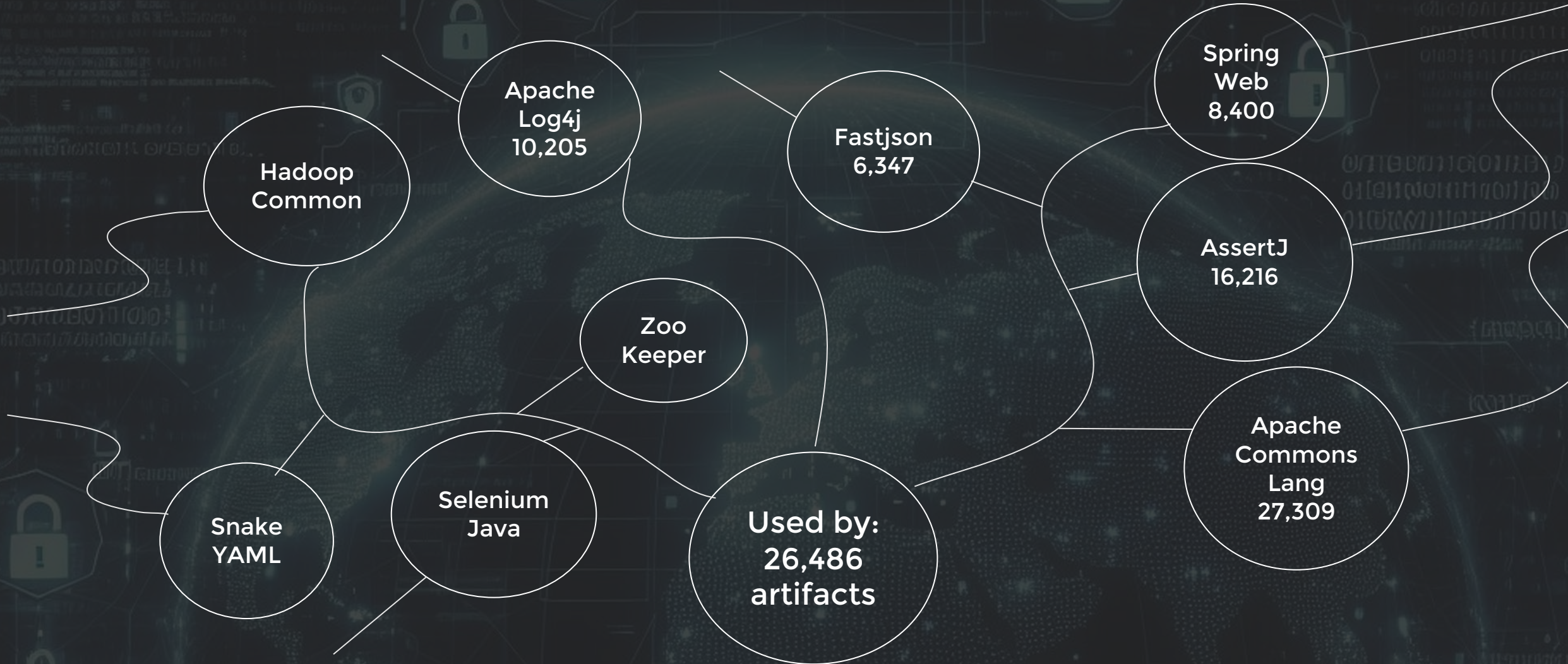
Freedom to deploy anywhere

Open source libraries are reused by other open source

Libraries can have up to 1000s of dependencies

Same issue with proprietary software since they are all using OSS

Example: Apache Commons IO Library



* Sources: graphcommons.com

* Source: MavenRepository.com (Nov 16, 2023)



Open Source Software Security

It's about Vulnerabilities

Common Vulnerability and Exposures (CVE)

Common Vulnerability Score System (CVSS)

Vulnerabilities outside CVE and the National Vulnerability Database (NVD)

New vulnerabilities are constantly being discovered

The smart way to make them public is when you have a fix (public disclosure)

More than 95% of public vulnerabilities have a fix, but you must keep up with the latest releases/patches

The Equifax Breach of 2017

EQUIFAX BREACH SETTLEMENT FUND

c/o JND Legal Administration
P.O. Box 91318
Seattle, WA 98111

Signature Bank
1-1357
260

No. 2737353
Date 01/17/2023

Five Dollars and Twenty-One cents.

\$5.21

Void after April 17, 2023

PAY TO THE
ORDER OF

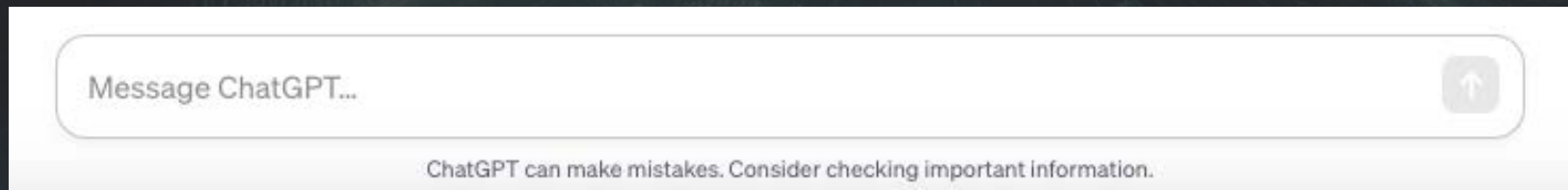
JAVIER PEREZ

Jerry J. W. Keesh

AUTHORIZED SIGNATURE

⑆ 2232353⑆ ⑆ 026013526⑆ 1501221115⑆

OpenAI's ChatGPT took the world by storm, over 100 million users in the first two months after its public launch



and there's more

 Copilot

 Bard

 HuggingChat

 Amazon Q

Prompt: Write basic code for user authentication in Python

ChatGPT

```
# Simple user authentication example in Python

# Sample user credentials (replace with your actual data)
user_credentials = {
    "user123": "password123",
    "john_doe": "securepass",
    "alice_wonder": "letmein"
}

def authenticate_user(username, password):
    """
    Authenticate the user with provided username and password.

    Args:
    - username (str): User's inputted username.
    - password (str): User's inputted password.

    Returns:
    - bool: True if authentication is successful, False otherwise.
    """
    # Check if the username exists in user_credentials, and if the provided password matches
    return username in user_credentials and user_credentials[username] == password

# Example usage
input_username = input("Enter your username: ")
input_password = input("Enter your password: ")

if authenticate_user(input_username, input_password):
    print("Authentication successful! Welcome,", input_username)
else:
    print("Authentication failed. Invalid username or password.")
```

Copilot

```
# Import Flask and Flask-Login
from flask import Flask, render_template, request, redirect, url_for
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required

# Create a Flask app and configure it
app = Flask(__name__)
app.config['SECRET_KEY'] = 'some-secret-key' # Change this to a secure value

# Define a User model that inherits from UserMixin
# UserMixin provides default implementations for some methods and properties
# You can also add your own attributes and methods as needed
class User(UserMixin):
    # A dictionary that maps user ids to user objects
    # In a real application, you would use a database instead
    user_database = {
        "user1": ("user1", "password1"),
        "user2": ("user2", "password2")
    }

    # A class method that returns a user object given a user id
    # Returns None if the user id is not found
    @classmethod
    def get(cls, user_id):
        user = cls.user_database.get(user_id)
        if user:
            return cls(user_id, user[1])
        return None

    # An instance method that initializes a user object with an id and a password
    def __init__(self, user_id, password):
        self.id = user_id
        self.password = password

# Create a login manager and initialize it with the app
login_manager = LoginManager()
login_manager.init_app(app)

# Define a callback function that loads a user object given a user id
# This is used by Flask-Login to retrieve the current user
@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id)
```

```
# Define a login view that handles the login form and authentication logic
@app.route('/login', methods=['GET', 'POST'])
def login():
    # If the request method is GET, render the login template
    if request.method == 'GET':
```

HuggingChat

```
class User:
    def __init__(self, username, hashed_password):
        self.username = username
        self.hashed_password = hashed_password

# A dictionary to store the users
USERS = {
    "user1": "$2b$12$qwertyuiopasdfghjkl".encode(),
    "user2": "$2b$12$ABCDEFGHIJKLMNQPQR".encode(),
}

def get_user(username):
    """Return the user object with given username if it exists."""
    if username in USERS:
        return User(username, USERS[username])
    else:
        return None

def register_user(username, plaintext_password):
    """Register a new user with given username and plaintext password."""
    hashed_password = "$2b$12$"+plaintext_password.encode().hex()+"$ "
    USERS[username] = hashed_password.encode()

def authenticate_user(username, plaintext_password):
    """Authenticate a user with given username and plaintext password."""
    user = get_user(username)
    if user and check_password(user.hashed_password, plaintext_password):
        return True
    else:
        return False

def check_password(hashed_password, plaintext_password):
    """Check whether the given plaintext password matches the hashed password
    prefix, salt, hashed = hashed_password.split('$')
    return bcrypt.checkpw(plaintext_password.encode(), bytes.fromhex(hashed))

if __name__ == '__main__':
    import sys
    import getpass
    import hashlib
```


CONCERN 1

Validity of the Code

Without context the validity of the code can be questioned.
Are developers going to rely entirely on AI-generated code?

RESPONSE

Developers don't take AI-generated code as the final word.

AI-generated code should be just the baseline.

Like using code from Stack Overflow, GitHub, NPM, etc.

CONCERN 2

Training Data

ML/DL models training must be fair, robust and explainable.
Where is the data coming from? Garbage in, garbage out.

RESPONSE

AI-generated code outputs are not guaranteed to be perfect.

But can explain new or existing code and provide unit tests to
produce better software and faster.

CONCERN 3

Ownership

Can I use and distribute AI-generated code?
What about license compliance?

RESPONSE

AI-generated code should not be considered in a final state or copyrightable.

It's not supposed to be identical to other code, if it is, Microsoft covers legal challenges related to copyright grounds (paid version)

CONCERN 4

Job Displacement

AI tools are going to make developer jobs redundant.

RESPONSE

Deeper testing shows inaccuracies, unreliable code, and content.

AI-generated code won't immediately create new or different jobs.

Encourage developers to explore ChatGPT, GitHub Copilot, Google Bard, Google Duet AI, Amazon Q, HuggingChat, and others.

CONCERN 5

Security

AI-generated code is going to introduce vulnerabilities.

RESPONSE

It's the developer's responsibility to use code ethically and safely.

It's important to carefully review any AI-generated code and ensure that it doesn't introduce vulnerabilities.

Same as it's done today from other sources.

AI tools have proven to be effective educational tools

Ask for book recommendations about programming languages and coding; it delivers short descriptions for each book

Prompt it for the top takeaways from one specific book

Translate code from one programming language to another

Other Security Concerns

- **Phishing attacks:** Generative AI allows bad foreign actors the ability to achieve near fluency in English
- **Malicious code:** AI is programmed not to generate code that it deems to be malicious. Code generation can still be manipulated

Open Source LLM Security

- Models are data, similar to binaries, can be inspected
- Models depend on the dataset, can be wrong, old, or abandoned
- Challenging or unavailable training steps and training data
- Models downloaded from Hugging Face or others



Other LLM Security Risks

- Prompt Injections to manipulate outputs
- Training Data Poisoning
- Data Leakage
- Inappropriate content or “Hallucinations”
- Insecure Plugins to external resources





If AI-generated code can assist bad actors, remember that this same power is equally available to good actors



Vulnerabilities in Open Source

Dev Security Education

- Security Champions Needed
- Knowledge of OWASP Top Ten

Many Contributors

“Given enough eyeballs, all bugs are shallow” – Linus’ Law



OWASP[®] Top 10

- Broken Access Control
- Cryptographic Failures (sensitive data exposure)
- Injection (SQL, NoSQL, OS, LDAP, Cross-site scripting)
- Insecure Design
- Security Misconfiguration
- Vulnerable and Outdated Components (CVEs and EOL software)
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging & Monitoring Failures
- Server-Side Request Forgery (SSRF)

Vulnerabilities Scanners

- Identify open source libraries, generate software bill of materials (SBOMs)
- Identify vulnerabilities and severity on libraries and dependencies
- Prioritization, alerts, and security policies
- Integrate with build environment (CI pipeline)

Open Source /Free Scanners

- OWASP Dependency-Checker
- NPM audit
- GitHub vulnerability alerts
- GitLab dependency scanning
- SAP code vulnerability analyzer

SBOMs Generation

- OWASP Dependency-Track
- SBOM Tool (Microsoft)
- Syft and Grype
- CycloneDx-CLI
- OSV-Scanner



Public Vulnerability Databases

National Vulnerability Database nvd.nist.gov

GitHub Advisory Database github.com/advisories

Mend Vulnerability Database

mend.io/vulnerability-database

NPM Security Advisory npmjs.com/advisories

Sonatype OSS Index ossindex.sonatype.org

VulDB vuldb.com

Metasploit rapid7.com/db

More Security Considerations

- End-of-Life Open Source Software
 - AngularJS
 - CentOS
 - PHP
 - others
- Untracked Dependencies
- Unknown Origin (Provenance)

“Open Source Security is about visibility of vulnerabilities, apply patches, avoid EOL versions, and making scans part of SDLC.”

**“Whether it’s reviewing code,
pair programming or learning
from fellow developers,
humans will not be replaced by
generative AI, only enhanced.”**

“Take advantage of the new tools and treat AI-generated code as any other code: with bugs and vulnerabilities.”

Images generated with OpenAI's DALL-E via Microsoft Bing Image Creator

Thank You!

Javier Perez

Perforce Software

 Chief Evangelist & Sr. Director of Product Management

 javierperez.mozello.com

 www.linkedin.com/in/javierperez