# ENHANCING TEST AUTOMATION AND SECURITY WITH PYTHON & AI FOR QUALITY-DRIVEN DEVSECOPS

SRIMAAN YARRAM

# ABOUT ME

Name: **Srimaan Yarram**

Current : **Sr Engineering Manager in Test**

Experience: 20+ years in Software Dev & Test Quality Engg

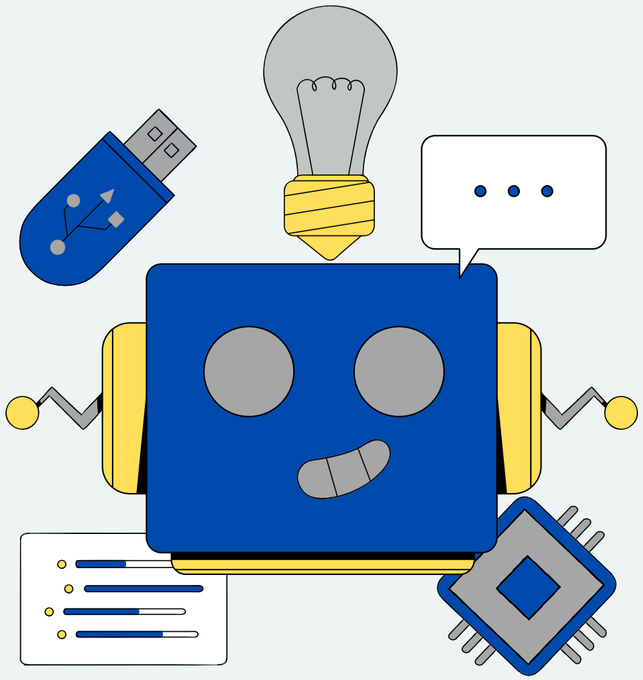
Expertise:

- AI/ML, Software Testing, DevSecOps
- Distributed Systems, Microservices, CI/CD
- Python, Java, AWS

LinkedIn: linkedin.com/in/srimaan


***Disclaimer: The views and content presented in this session are solely my own and do not represent those of my employer or any affiliated organizations. Some of the code provided is for demonstration purposes only and may not be fully functional or executable as presented. It is intended to illustrate concepts rather than serve as production-ready code.***

# AGENDA

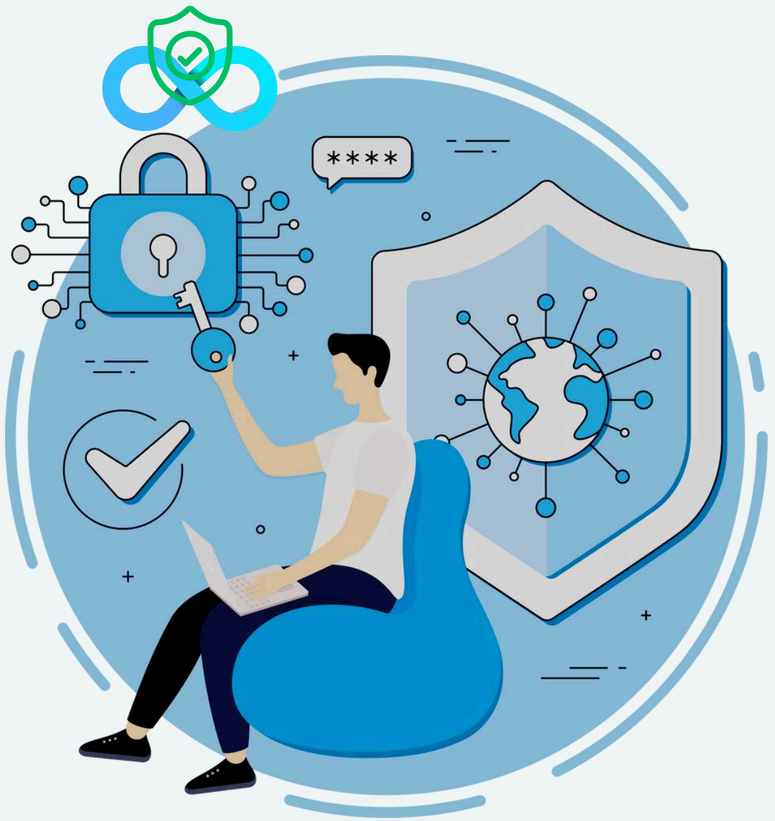


- Introduction & DevOps vs. **DevSecOps**
- Role of Python in DevSecOps
- Leveraging AI in DevSecOps
- Tool-Based Approach to DevSecOps
  - Static Code Analysis (Dev Phase)
  - Secrets Detection (Code Review Phase)
  - Dependency Scan (Build Phase)
  - Functional & Security Testing (Deployment Phase)
  - Post-Deployment Monitoring (Production Phase)
- Best Practices

# INTRODUCTION

**DevOps**:
Focuses on speed & collaboration between Dev and Ops.

**DevSecOps**:
Embeds security throughout the development lifecycle.




## WHY THE SHIFT?

EVOLVING THREATS DEMAND PROACTIVE SECURITY INTEGRATION.

# ROLE OF PYTHON IN DEVSECOPS

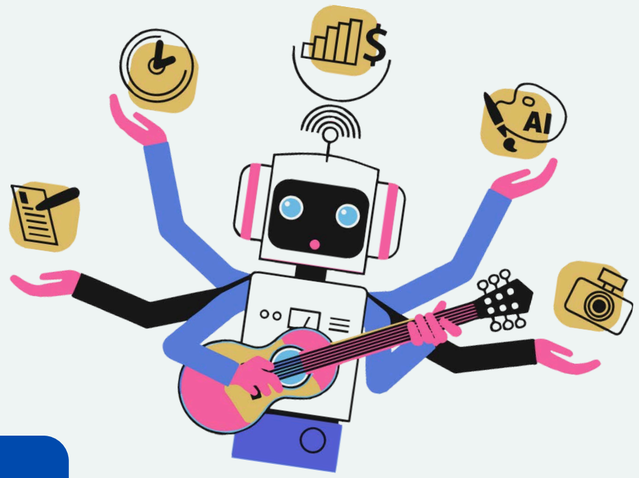

## WHY PYTHON?

### SIMPLICITY, FLEXIBILITY, RICH ECHO SYSTEM

**Security Automation:**

- Vulnerability Scanning
- Compliance Checks, and
- Threat Detection

# ROLE OF AI IN SECURITY AND DEVSECOPS




## 01

### WHY AI IN SECURITY?

PREDICT

DETECT

RESPOND

## 02

### WHAT ARE THE KEY BENEFITS

AUTOMATED THREAT DETECTION,

INTELLIGENT CODE ANALYSIS,

CONTINUOUS SECURITY IMPROVEMENT

## 03

### HOW DOES AI CONTRIBUTE TO DEVSECOPS

SMARTER TESTING PIPELINES,

REAL-TIME ANOMALY DETECTION,

AUTOMATED COMPLIANCE CHECKS

AI IN CYBERSECURITY IS PROJECTED TO REACH $46B BY 2027, REFLECTING ITS GROWING IMPACT ON MODERN SECURITY FRAMEWORKS

# TOOL-BASED AI INTEGRATION IN DEVSECOPS

## STATIC ANALYSIS - DEV PHASE

Pylint
CodeQL

## SECRETS DETECTION - CODE REVIEW PHASE

GitLeaks
TruffleHog

## DEPENDENCY SCANNING - BUILD PHASE

Safety,
Dependabot


## FUNCTIONAL TESTING - DEPLOYMENT PHASE

Postman
Healenium

## SECURITY TESTING - POST DEPLOYMENT

OWASP ZAP


## ANOMALY DETECTION - PRODUCTION PHASE

Loguru
AI Modles

# DEV PHASE

## Code Quality and Security Issues

- **PyLint,**
- **CodeQL**

**AI –** Enhancing analysis with AI

- Automated code improvements
- Context-aware vulnerability detection
- Adaptive code style enforcement

```
1 usage
def run_pylint():
    result = subprocess.run( args: ['pylint', 'flask_vulnerable_app.py'], capture_output=True, text=True)
    print("Pylint Output:\n", result.stdout)
    return result.stdout

1 usage
def run_codeql():
    subprocess.run(['codeql', 'database', 'create', 'flask-db', '--language=python', '--source-root=.'])
    result = subprocess.run( args: ['codeql', 'database', 'analyze', 'flask-db', 'python-code-scanning.qls',
                                    '--format=sarif-latest'], capture_output=True, text=True)
    print("CodeQL Output:\n", result.stdout)
    return result.stdout

2 usages
def extract_issues_with_line_numbers(output):...

1 usage
def use_codet5_for_fixes(issue_text):
    model_name = 'Salesforce/codet5-base'
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

    inputs = tokenizer.encode(issue_text, return_tensors="pt")
    outputs = model.generate(inputs)
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

if __name__ == '__main__':...
```

```
==== Pylint Analysis Report ====
Line 7: Constant name 'SECRET_KEY' doesn't conform to UPPER_CASE naming style.
Line 15: Command injection detected via os.popen.
==== End of Pylint Report ====

==== AI Suggested Fixes for Pylint Issues ====
Line 7: Replace 'SECRET_KEY = hardcoded_secret' with 'SECRET_KEY = os.getenv("SECRET_KEY")'
Line 15: Use 'subprocess.run(shlex.split(command), check=True)' instead of os.popen.
```

```
==== CodeQL Security Analysis Report ====
Line 22: SQL Injection vulnerability detected in raw query.
Line 30: Cross-site scripting (XSS) vulnerability in user input.
==== End of CodeQL Report ====

==== AI Suggested Fixes for CodeQL Issues ====
Line 22: Replace raw query with parameterized SQL query:
cursor.execute("SELECT * FROM users WHERE username=? AND password=?", (username, password))
Line 30: Use 'escape(name)' to sanitize user input in HTML rendering.
==== End of AI Fixes ====
```

# REVIEW PHASE

Hardcoded credentials, API keys, and sensitive data

**GitLeaks**

**TruffleHog**

**AI Integration:**

- Reducing false positives
- Smart remediation suggestions
- Continuous Improvement

```python
TARGET_REPO = "."

2 usages
def run_scan(tool, args):
    try:
        result = subprocess.run([tool] + args, capture_output=True, text=True, check=True)
        print(f"==== {tool} Report (Before AI) ====\n", result.stdout)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print(f"Error running {tool}:", e)
        return ""

1 usage
def run_ai_suggestions(th_output, gl_output):
    print("\n==== AI Integration (After AI) ====")
    model = AutoModelForSeq2SeqLM.from_pretrained('Salesforce/codet5-base')
    tokenizer = AutoTokenizer.from_pretrained('Salesforce/codet5-base')
    inputs = tokenizer.encode(th_output + "\n" + gl_output, return_tensors="pt")
    suggestions = tokenizer.decode(model.generate(inputs, max_length=512)[0], skip_special_tokens=True)
    print("AI Suggested Fixes:\n", suggestions)

if __name__ == "__main__":
    th_output = run_scan( tool: "trufflehog",  args: ["filesystem", TARGET_REPO, "--json"])
    gl_output = run_scan( tool: "gitleaks",  args: ["detect", "--source", TARGET_REPO, "--verbose"])
    run_ai_suggestions(th_output, gl_output)
```

## TRUFFLEHOG

```
==== TruffleHog Report (Before AI) ====
{
  "path": "config.py",
  "stringsFound": [
    "aws_access_key = 'AKIA123456789EXAMPLE'",
    "password = 'mypassword123'"
  ]
}
```

```
==== AI Integration (After AI) ====
File: config.py, Line: 2
Issue: Hardcoded AWS access key detected.
AI Suggested Fix: Move the key to AWS Secrets
Manager and reference it via an environment variable.
```

## GITLEAKS

```
==== GitLeaks Report (Before AI) ====
{
  "file": "secrets.env",
  "line": 10,
  "match": "DB_PASSWORD=supersecret123",
  "rule": "Generic Credential"
}
```

```
==== AI Integration (After AI) ====
File: secrets.env, Line: 10
Issue: Hardcoded database password detected.
AI Suggested Fix: Use environment variables or a secure
secrets manager instead of storing credentials in plain text.
```

# BUILD PHASE

**Hidden vulnerabilities in third-party libraries**

- **Safety,**
- **Dependabot**

**AI Integration**

- Prioritization of vulnerabilities
- Intelligent remediation strategies

```python
# Step 1: Scan dependencies for vulnerabilities using Safety
def scan_dependencies():
    result = subprocess.run(["safety", "check", "--json"],
                            capture_output=True, text=True)
    return json.loads(result.stdout)

dependency_vulnerabilities = scan_dependencies()

# Step 2: Analyze code for deprecated function usage
def analyze_code_for_deprecations(file_path):
    with open(file_path, 'r') as file:
        tree = ast.parse(file.read())
        for node in ast.walk(tree):
            if isinstance(node, ast.Call) and hasattr(node.func, 'attr'):
                print(f"Potential deprecated function '{node.func.attr}'
                found in {file_path} at line {node.lineno}")

analyze_code_for_deprecations("example_code.py")
```

```python
# Step 3: Visualize dependency vulnerabilities using NetworkX
graph = nx.Graph()
for dep in dependency_vulnerabilities:
    graph.add_node(dep.get("package_name"))
    graph.add_edge(dep.get("package_name"), dep.get("cve") or "Unknown")

nx.draw(graph, with_labels=True, node_color="lightblue", edge_color="red")
plt.title("Dependency Vulnerability Graph")
plt.show()

# Step 4: AI-based risk prediction using PyTorch Geometric
class RiskModel(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1, self.conv2 = GCNConv(3, 16), GCNConv(16, 1)

    def forward(self, x, edge_index):
        return self.conv2(self.conv1(x, edge_index).relu(), edge_index)

model = RiskModel()
x = torch.rand((len(dependency_vulnerabilities), 3))
edge_index = torch.tensor([[i, i+1] for i in range(len(dependency_vulnerabilities)-1)], dtype=torch.long).T
print("Predicted Risk Scores:", model(x, edge_index).detach().numpy())
```

```
43
44    # Step 5: Fetch real-time security advisories from GitHub
45    g = Github("your_github_token")
46    alerts = g.get_repo("owner/repository").get_vulnerability_alerts()
47    for alert in alerts:
48        print(f"Dependency: {alert.affected_package_name}, Severity: {alert.severity}")
49
```

## BEFORE AI (MANUAL PROCESS):

```
Dependency: numpy, CVE: CVE-2021-1234, Severity: High
Dependency: requests, CVE: CVE-2020-5678, Severity: Medium
Potential deprecated function 'old_function' found in example_code.py at line 12.
```

## AFTER AI (AUTOMATED PROCESS)

```
Dependency: numpy, CVE: CVE-2021-1234, Severity: High, Risk Score: 0.85
Recommended Action: Upgrade to numpy>=1.21.0
Impact Analysis: Deprecated functions detected in 'example_code.py'
Suggested Fix: Replace old_function with new_function.

Dependency: requests, CVE: CVE-2020-5678, Severity: Medium, Risk Score: 0.65
Recommended Action: Review API usage and apply patches.
Impact Analysis: No breaking changes detected.
```

# FUNCTIONAL TEST

Feature correctness & reliability
Early defect identification (APIs & UI)

Postman, Newman
Healenium

**AI Integration Benefits**

- Smart test prioritization
- Self-healing automation
- Actionable insights

```python
# Step 1: Setup WebDriver with Healenium for self-healing
options = webdriver.ChromeOptions()
options.add_argument("--headless")
driver = webdriver.Remote(command_executor="http://localhost:4444/wd/hub", options=options)

# Initialize Healenium for tracking changes
healenium = Healenium(driver)

def run_ui_test():
    driver.get("https://example.com/login")
    element = healenium.find_element_by_xpath("//input[@id='username']")
    element.send_keys("test_user")
    driver.find_element_by_xpath("//input[@id='password']").send_keys("securePass")
    driver.find_element_by_xpath("//button[@id='login']").click()
    assert "Dashboard" in driver.title
    print("UI Test Passed")

run_ui_test()
driver.quit()

# Step 2: API Testing with Postman-like functionality
def api_test():
    response = requests.get("https://jsonplaceholder.typicode.com/posts/1")
    assert response.status_code == 200
    assert response.json()['id'] == 1
    print("API Test Passed")

api_test()
```

# SECURITY TEST

Preventing vulnerabilities before production
Protecting sensitive data and compliance

OWASP ZAP, Nikto

**AI Integration Benefits**

Automated threat detection
Real-time vulnerability analysis
Smart remediation suggestions

```python
# Step 1: Run OWASP ZAP and Nikto Scans
def run_security_scans(target_url):
    subprocess.run(f"nikto -h {target_url} -output nikto_scan.json -Format json", shell=True)
    subprocess.run(f"zap-cli quick-scan {target_url}", shell=True)
    with open("nikto_scan.json") as f:
        nikto_results = json.load(f)
    return {"nikto": nikto_results, "zap": get_zap_results()}

# Step 2: Get ZAP Scan Results
def get_zap_results():
    return json.loads(subprocess.check_output(["zap-cli", "alerts", "-f", "json"]))

# Step 3: Prioritize Findings
def prioritize_findings(scan_results):
    clf = RandomForestClassifier().fit([[3, 50], [2, 40]], [1, 0])
    for tool, results in scan_results.items():
        for alert in results.get("alerts", []):
            alert['priority'] = 'High' if clf.predict([[alert['risk'],
             len(alert['url'])]])[0] else 'Low'
    return scan_results

# Step 4: Reduce False Positives
def filter_false_positives(scan_results):
    nlp_model = pipeline("text-classification", model="distilbert-base-uncased")
    for tool, results in scan_results.items():
        scan_results[tool]['alerts'] = [alert for alert in results.get("alerts", [])
           if nlp_model(alert["description"])[0]["label"] != "benign"]
    return scan_results

# Step 5: Suggest Fixes
def suggest_fixes(scan_results):…

# Step 6: Run Security Testing
def process_security_testing(target_url):…

if __name__ == "__main__":
    process_security_testing("http://srimaan.tech")
```

# BEFORE AI INTEGRATION

```
OWASP ZAP Security Alerts:
[
  {"name": "XSS", "url": "http://example.com/contact", "risk": "High"},
  {"name": "SQL Injection", "url": "http://example.com/login", "risk": "Medium"},
  {"name": "Missing Security Headers", "url": "http://example.com/home", "risk": "Low"}
]

Nikto Security Report:
- The web server is outdated.
- X-Content-Type-Options header is missing.
- Possible SQL Injection vulnerabilities detected.
```

# AFTER AI INTEGRATION:

```
AI-Powered Security Insights:

Predicted Risk Score: 0.92 (Critical)

Prioritized Vulnerabilities:
1. SQL Injection at http://example.com/login (Risk Score: 9.5/10)
   Suggested Fix: Use parameterized queries and input validation.

2. XSS at http://example.com/contact (Risk Score: 8.3/10)
   Suggested Fix: Implement proper input sanitization using a security library.

3. Missing Security Headers at http://example.com/home (Risk Score: 4.1/10)
   Suggested Fix: Add security headers like Content-Security-Policy and X-Frame-Options.

AI-Generated Remediation Plan:
"To fix the SQL Injection vulnerability, apply input validation and use prepared statement

Threat Prediction:
Potential RCE (Remote Code Execution) risks detected based on historical data.
```

# AI-POWERED LOG AND ANOMALY DETECTION

Detecting patterns and potential security threats
Reducing mean time to detect (MTTD) and mean time to respond (MTTR)

Loguru,
Elastic Stack,

**AI Integration Benefits**
**Predictive anomaly detection**
**Automated log pattern recognition**
**Real-time alerts and insights**

```python
# Step 1: Log Processing Stub
def process_logs(log_file):
    """
    Read and process log data from the given file.
    """
    pass  # Implement log reading and preprocessing logic

# Step 2: AI-Based Anomaly Detection Stub
def detect_anomalies(logs):
    """
    Use AI models to detect anomalies in log data.
    """
    pass  # Implement AI-based anomaly detection

# Step 3: Risk Prediction Stub
def predict_risk(anomalies):
    """
    Predict risk levels of identified anomalies using AI models.
    """
    pass  # Implement risk prediction logic

# Step 4: Smart Containment Stub
def smart_containment(anomalies):
    """
    Take containment actions based on risk assessment.
    """
    pass  # Implement actions such as blocking or monitoring

# Step 5: Execution Stub
def run_security_analysis(log_file):
    """
    Execute the full security analysis workflow.
    """
    pass  # Implement the execution flow combining all steps

if __name__ == "__main__":
    run_security_analysis("log_data.json")
```

## EHAMPLE LOGS (BEFORE AI):

```
[INFO]  2024-01-24 10:00:01  - User login attempt from IP 192.168.1.10
[WARNING]  2024-01-24 10:02:15  - Multiple failed login attempts detected
[ERROR]  2024-01-24 10:03:45  - Potential brute force attack detected
```

## EHAMPLE LOGS (AFTER AI):

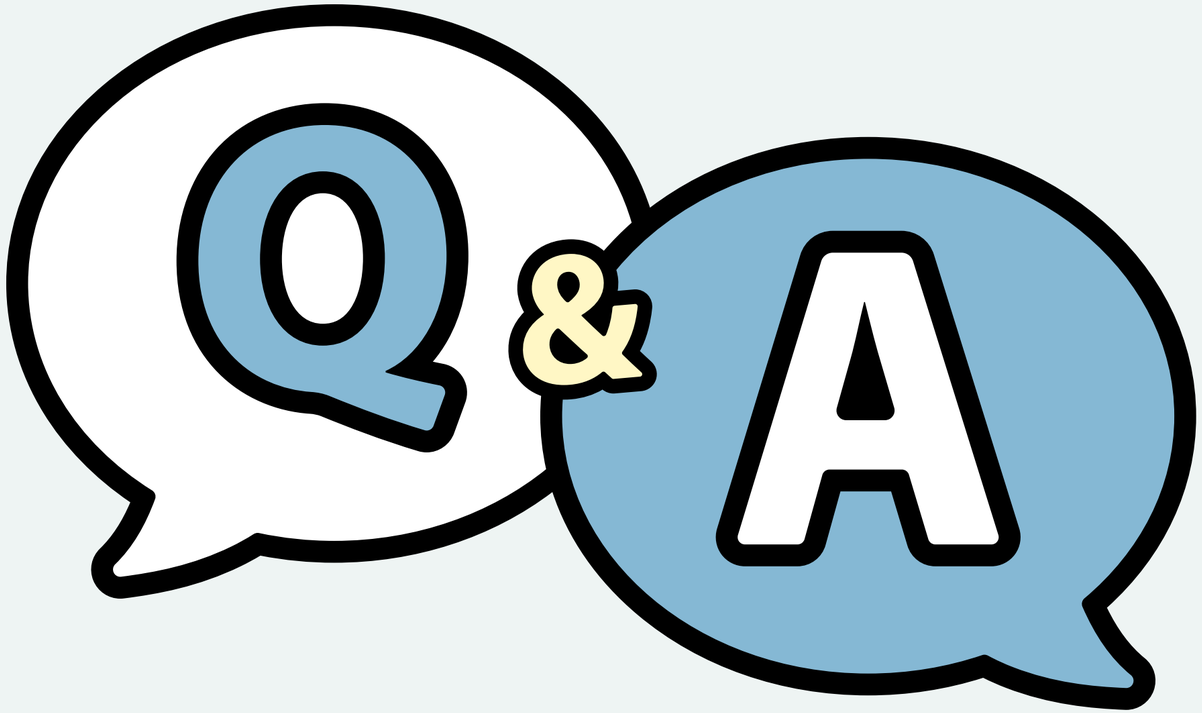
```
[ALERT] Brute Force Attack Detected on http://fakeecommerce.com/login
Risk Level: Critical (Score: 9.5/10)
Action Taken:
- IP 192.168.1.10 BLOCKED
- MFA enabled for affected accounts
- Session tokens revoked
Suggested Fix: Implement rate limiting and CAPTCHA
```

# BEST PRACTICES FOR AI-DRIVEN DEVSECOPS

- Start Small, Grow Smart
- Prioritize What Matters.
- Keep It Transparent.
- Always Improve
- Integrate Smoothly

# THE FUTURE OF AI IN DEVSECOPS

- AI enhances speed, accuracy, and efficiency.
- Provides proactive security and automation.
- Scales testing efforts without human bias.
- Enables smarter decision-making through insights.
- Continuous improvement with feedback loops.


Q&A