🔐 Lesson 6 AI for Vulnerability Assessment - Student Code Exercise Guide

📋 Prerequisites

Required Software

```
# Install required Python packages
pip install pandas numpy scikit-learn matplotlib seaborn requests
beautifulsoup4 jupyter
# Optional: For advanced exercises
pip install plotly dash streamlit
```

Knowledge Requirements

- Basic Python programming
- Understanding of machine learning concepts (from Classes 2-3)
- Familiarity with cybersecurity terminology

🎯 Learning Objectives

By completing these exercises, you will:

1. **Apply ML to real vulnerability data**
2. **Build practical vulnerability prioritization systems**
3. **Implement anomaly detection for security**
4. **Understand the challenges of AI in cybersecurity**

📊 Exercise 1: Vulnerability Risk Prioritization

Objective: Build an AI system to prioritize vulnerability remediation

Task Overview

1. Run the vulnerability prioritization code
2. Experiment with different features

3. Compare ML predictions to traditional CVSS scoring

Step-by-Step Instructions

Step 1: Generate and Explore Data

```
# Run the vulnerability dataset creation
vuln_df = create_vulnerability_dataset()
# YOUR TASK: Analyze the data distribution
print("Data Summary:")
print(vuln_df.describe())
# Create visualizations
import matplotlib.pyplot as plt
import seaborn as sns
# Plot CVSS score distribution
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.hist(vuln_df['cvss_score'], bins=20, alpha=0.7)
plt.title('CVSS Score Distribution')
plt.xlabel('CVSS Score')
plt.subplot(1, 3, 2)
sns.countplot(data=vuln_df, x='asset_criticality')
plt.title('Asset Criticality Distribution')
plt.xticks(rotation=45)
plt.subplot(1, 3, 3)
sns.boxplot(data=vuln_df, x='high_priority', y='cvss_score')
plt.title('CVSS vs Priority Classification')
plt.tight_layout()
plt.show()
```

Step 2: Feature Engineering Challenge

```
# YOUR CHALLENGE: Create new features that might improve prediction
def engineer_new_features(df):
    """
    Add your own feature engineering here!
    Ideas:
    - Risk velocity: vulnerability_age_days / system_uptime_days
    - Attack surface: internet_facing * affected_systems
    - Patch urgency: exploit_available * (1 - patch_available)
    """

    df_new = df.copy()
```

```
    # Example new features (you can add more!)
    df_new['risk_velocity'] = df['vulnerability_age_days'] /
(df['system_uptime_days'] + 1)
    df_new['attack_surface'] = df['internet_facing'] *
df['affected_systems']
    df_new['patch_urgency'] = df['exploit_available'] * (1 -
df['patch_available'])

    # YOUR TURN: Add 2-3 more features here
    # Feature 1:
    # Feature 2:
    # Feature 3:

    return df_new

# Test your feature engineering
vuln_enhanced = engineer_new_features(vuln_df)
print("New features added:", set(vuln_enhanced.columns) -
set(vuln_df.columns))
```

Step 3: Model Comparison

```
# YOUR TASK: Compare different ML algorithms

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Prepare your enhanced features
X_enhanced, y = prepare_vulnerability_features(vuln_enhanced)
X_train, X_test, y_train, y_test = train_test_split(X_enhanced, y,
test_size=0.2, random_state=42)

# Test multiple algorithms

models = {
    'Random Forest': RandomForestClassifier(n_estimators=100,
random_state=42),
    'Logistic Regression': LogisticRegression(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'SVM': SVC(random_state=42, probability=True)
```

```
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)


    results[name] = {
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred)
    }

# Display results

results_df = pd.DataFrame(results).T
print("Model Comparison:")
print(results_df.round(3))
```

Questions for Reflection

1. Which features are most important for vulnerability prioritization?
2. How does your enhanced model compare to the baseline?
3. What would happen if you only used CVSS scores?
4. How might false positives impact security teams?

🔮 Exercise 2: Vulnerability Prediction

Objective: Predict vulnerability likelihood in software projects

Extended Challenge

```
# YOUR CHALLENGE: Improve the vulnerability prediction model

def create_enhanced_software_dataset():
    """
    Enhance the software dataset with additional realistic features
    """
    df = create_software_vulnerability_dataset()
```

```
    # Add more realistic features
    df['git_commits_per_month'] = np.random.poisson(50, len(df))
    df['test_coverage'] = np.random.beta(2, 3, len(df))  # Typically low
    df['static_analysis_violations'] = np.random.poisson(10, len(df))
    df['previous_vulnerabilities'] = np.random.poisson(2, len(df))

    # YOUR TASK: Add domain-specific features

    # Consider: team size, deployment frequency, update cadence, etc.

    return df

# Test your enhanced dataset

enhanced_software = create_enhanced_software_dataset()

# Train and evaluate your improved model

# Compare against the baseline model
```

Real-World Data Integration

```
# ADVANCED: Try with real vulnerability data

# You can download CVE data from NIST NVD or use vulnerability databases

def load_real_cve_data():
    """
    Example function to work with real CVE data
    Note: You'll need to download actual CVE data for this to work
    """
    try:
        # Example URL - replace with actual data source
        # df = pd.read_json('path_to_cve_data.json')
        print("Real CVE data integration - implement based on available
data sources")
        print("Suggested sources:")
        print("- NIST NVD: https://nvd.nist.gov/vuln/data-feeds")
        print("- CVE Details: https://www.cvedetails.com/")
        print("- Vulnerability databases from security vendors")
```

```
    except Exception as e:
        print(f"Using synthetic data. For real data: {e}")
load_real_cve_data()
```

🚨 Exercise 3: Security Anomaly Detection

Objective: Detect unusual patterns that might indicate security incidents

Advanced Anomaly Detection Challenge

```
# YOUR CHALLENGE: Improve anomaly detection accuracy

def create_realistic_network_data():
    """
    Create more realistic network traffic with time-based patterns
    """
    np.random.seed(42)
    n_records = 5000

    # Generate timestamps (24 hours of data)
    timestamps = pd.date_range('2024-01-01', periods=n_records, freq='30S')

    # Create time-based patterns (higher traffic during business hours)
    hours = timestamps.hour
    business_hour_multiplier = np.where((hours >= 9) & (hours <= 17), 1.5,
0.5)

    # YOUR TASK: Create more sophisticated traffic patterns
    # Consider: weekend vs weekday, lunch breaks, night shifts, etc.

    normal_traffic = {
        'timestamp': timestamps,
        'bytes_transferred': np.random.lognormal(10, 1, n_records) *
business_hour_multiplier,
        'connection_count': np.random.poisson(20, n_records) *
business_hour_multiplier,
        'unique_ips': np.random.poisson(10, n_records),
        'port_variety': np.random.poisson(5, n_records),
        'protocol_diversity': np.random.gamma(2, 2, n_records),
```

```
        'avg_packet_size': np.random.normal(1500, 200, n_records),
        'connection_duration': np.random.exponential(60, n_records)
    }

    df = pd.DataFrame(normal_traffic)

    # Inject sophisticated anomalies
    anomaly_indices = np.random.choice(len(df), size=int(len(df) * 0.05),
replace=False)

    # YOUR TASK: Create different types of anomalies

    # 1. DDoS-like patterns

    # 2. Data exfiltration patterns

    # 3. Port scanning patterns

    # 4. Insider threat patterns

    return df, anomaly_indices

# Implement and test your enhanced anomaly detection
enhanced_traffic, anomaly_indices = create_realistic_network_data()
```

Multi-Algorithm Comparison

```
# YOUR TASK: Compare different anomaly detection algorithms

from sklearn.ensemble import IsolationForest
from sklearn.cluster import DBSCAN
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor

# Implement and compare:

# 1. Isolation Forest

# 2. DBSCAN clustering

# 3. Elliptic Envelope
```

```
# 4. Local Outlier Factor

def compare_anomaly_algorithms(X, true_anomalies):
    """
    Compare different anomaly detection algorithms
    """
    algorithms = {
        'Isolation Forest': IsolationForest(contamination=0.05,
random_state=42),
        'DBSCAN': DBSCAN(eps=0.3, min_samples=10),
        'Elliptic Envelope': EllipticEnvelope(contamination=0.05,
random_state=42),
        'LOF': LocalOutlierFactor(n_neighbors=20, contamination=0.05)
    }


    results = {}

    # YOUR IMPLEMENTATION HERE

    return results

# Test your comparison function
```

🔬 Exercise 4: Smart Fuzzing Simulation

Objective: Simulate intelligent vulnerability discovery

Advanced Fuzzing Challenge

```
# YOUR CHALLENGE: Create a more sophisticated fuzzing simulation

class SmartFuzzer:
    def __init__(self):
        self.vulnerability_patterns = {
            'buffer_overflow': {'complexity': 3, 'detection_rate': 0.15},
            'sql_injection': {'complexity': 2, 'detection_rate': 0.08},
            'xss': {'complexity': 2, 'detection_rate': 0.12},
            'path_traversal': {'complexity': 1, 'detection_rate': 0.10},
```

```
            'command_injection': {'complexity': 3, 'detection_rate': 0.06}
        }

    def adaptive_fuzzing(self, target_type, learning_rounds=10):
        """
        Simulate adaptive fuzzing that learns from previous attempts
        """
        success_rate =
self.vulnerability_patterns[target_type]['detection_rate']

        results = []
        for round_num in range(learning_rounds):
            # Simulate learning - success rate improves over time
            adaptive_rate = success_rate * (1 + round_num * 0.1)
            adaptive_rate = min(adaptive_rate, 0.5)  # Cap at 50%

            tests_this_round = np.random.randint(50, 200)
            successes = np.random.binomial(tests_this_round, adaptive_rate)

            results.append({
                'round': round_num + 1,
                'tests': tests_this_round,
                'successes': successes,
                'success_rate': successes / tests_this_round,
                'cumulative_rate': adaptive_rate
            })


        return pd.DataFrame(results)


    # YOUR TASK: Implement additional methods

    # - mutation_strategy()

    # - coverage_guided_fuzzing()

    # - ai_guided_input_generation()

# Test your smart fuzzer

fuzzer = SmartFuzzer()
fuzzing_results = fuzzer.adaptive_fuzzing('buffer_overflow')
```

```
print(fuzzing_results)

# Visualize learning curve
plt.figure(figsize=(10, 6))
plt.plot(fuzzing_results['round'], fuzzing_results['success_rate'],
marker='o', label='Actual Success Rate')
plt.plot(fuzzing_results['round'], fuzzing_results['cumulative_rate'],
marker='s', label='Expected Rate')
plt.xlabel('Fuzzing Round')
plt.ylabel('Success Rate')
plt.title('Smart Fuzzing Learning Curve')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

🏆 Exercise 5: Comprehensive Security Dashboard

Objective: Build an integrated vulnerability management dashboard

Final Integration Challenge

```
# YOUR FINAL CHALLENGE: Create a comprehensive security dashboard

def create_security_dashboard():
    """
    Integrate all components into a unified security assessment system
    """

    # 1. Load and process vulnerability data
    vuln_data = create_vulnerability_dataset()

    # 2. Run ML models for prioritization
    X, y = prepare_vulnerability_features(vuln_data)
    priority_model = RandomForestClassifier(n_estimators=100,
random_state=42)
    priority_model.fit(X, y)

    # 3. Generate risk scores
    risk_scores = priority_model.predict_proba(X)[:, 1]
```

```
    vuln_data['ml_risk_score'] = risk_scores

    # 4. Create summary metrics
    dashboard_metrics = {
        'total_vulnerabilities': len(vuln_data),
        'high_priority_count': (risk_scores > 0.7).sum(),
        'critical_assets_affected':
vuln_data[vuln_data['asset_criticality'] == 'Critical'].shape[0],
        'internet_facing_vulns': vuln_data['internet_facing'].sum(),
        'avg_cvss_score': vuln_data['cvss_score'].mean(),
        'avg_ml_risk_score': risk_scores.mean()
    }

    # 5. YOUR TASK: Add more sophisticated analytics

    # - Trend analysis

    # - Risk heat maps

    # - Remediation recommendations

    # - Cost-benefit analysis


    return vuln_data, dashboard_metrics

# Implement your dashboard

dashboard_data, metrics = create_security_dashboard()
print("Security Dashboard Metrics:")
for key, value in metrics.items():
    print(f"{key}: {value:.2f}")
```

Optional: Interactive Dashboard

BONUS: Create an interactive dashboard with Streamlit or Dash

```
# This is for advanced students interested in web development
"""
Example Streamlit app structure:
import streamlit as st
st.title("🔐 AI-Powered Vulnerability Assessment Dashboard")
```

```
# Sidebar controls
st.sidebar.header("Dashboard Controls")
risk_threshold = st.sidebar.slider("Risk Threshold", 0.0, 1.0, 0.7)
asset_filter = st.sidebar.multiselect("Asset Types", ['Low', 'Medium',
'High', 'Critical'])
# Main dashboard
col1, col2, col3 = st.columns(3)
with col1:
    st.metric("Total Vulnerabilities", total_vulns)

with col2:

    st.metric("High Priority", high_priority_count)

with col3:
    st.metric("Avg Risk Score", avg_risk_score)

# Charts and visualizations
st.plotly_chart(create_risk_distribution_chart())
st.plotly_chart(create_priority_matrix())

# Run with: streamlit run dashboard.py

"""
```

📝 Assessment Questions

Knowledge Check Questions

1. **Conceptual Understanding**

    - What are the key differences between vulnerability scanning and penetration testing?
    - How does AI improve traditional vulnerability assessment methods?
    - What are the main challenges in implementing AI for cybersecurity?

2. **Technical Implementation**

    - Explain why Random Forest works well for vulnerability prioritization
    - How would you handle imbalanced datasets in security applications?
    - What metrics are most important for evaluating anomaly detection systems?

3. **Real-World Application**

    - How would you convince a security team to adopt AI-powered vulnerability management?
    - What are the risks of over-relying on automated vulnerability assessment?
    - How should AI complement human expertise in cybersecurity?

Practical Assignments

1. **Feature Engineering Project**

    - Create 5 new features for vulnerability prioritization
    - Justify your choices with domain knowledge
    - Measure their impact on model performance

2. **Anomaly Detection Evaluation**

    - Implement 3 different anomaly detection algorithms
    - Compare their performance on network security data
    - Analyze trade-offs between false positives and detection rate

3. **Real-World Integration**

    - Research existing vulnerability management tools
    - Propose how AI could enhance one specific tool
    - Create a simple proof-of-concept implementation

🛠️ Additional Resources

Datasets for Further Practice

- **NVD (National Vulnerability Database)**: https://nvd.nist.gov/
- **CVE Details**: https://www.cvedetails.com/
- **OWASP WebGoat**: https://owasp.org/www-project-webgoat/
- **KDD Cup 1999**: Network intrusion detection dataset
- **NSL-KDD**: Updated version of KDD Cup 1999

Tools to Explore

- **Vulnerability Scanners**: OpenVAS, Nessus, Qualys
- **Penetration Testing**: Metasploit, Burp Suite, OWASP ZAP
- **AI/ML Libraries**: scikit-learn, TensorFlow, PyTorch
- **Security Analytics**: Splunk, ELK Stack, Suricata

Research Papers

- "Machine Learning for Vulnerability Assessment" (IEEE Security & Privacy)
- "AI-Powered Penetration Testing" (ACM CCS)
- "Anomaly Detection in Cybersecurity" (USENIX Security)

Industry Reports

- Gartner Magic Quadrant for Vulnerability Assessment
- SANS Vulnerability Management Survey
- Verizon Data Breach Investigations Report

🎯 Success Criteria

By completing these exercises, you should be able to:

✅ **Build ML models** for vulnerability prioritization
✅ **Implement anomaly detection** for security monitoring
✅ **Evaluate and compare** different AI approaches
✅ **Understand trade-offs** between accuracy and practicality
✅ **Apply domain knowledge** to improve model performance
✅ **Communicate results** to non-technical stakeholders

💡 Tips for Success

1. **Start Simple**: Begin with basic models before adding complexity
2. **Validate Assumptions**: Use domain knowledge to sanity-check results
3. **Focus on Practicality**: Consider how models would work in real environments
4. **Iterate Rapidly**: Try many approaches quickly rather than perfecting one
5. **Document Everything**: Keep notes on what works and what doesn't
6. **Seek Feedback**: Discuss results with peers and instructors

Good luck with your vulnerability assessment AI journey! Remember: the goal is not just to build models, but to understand how AI can make cybersecurity more effective and efficient.