# DAST Student Worksheet - Dynamic Application Security Testing

**Name:** **\_\_\_\_** **Date:** **\_\_\_\_**

**Lab Partner:** **\_\_\_\_** **Section:** **\_\_\_\_**

## � Pre-Exercise Setup Verification

**Before starting the DAST analysis, verify your environment is ready:**

### Step 1: Check Docker Containers

# Command to run:  
cd docker && docker-compose ps

**Expected Output**: You should see containers running including cybersec\_sandbox and vulnerable\_web\_app

**Container Status Check:**

* ☐ cybersec\_sandbox - State: Up
* ☐ vulnerable\_web\_app - State: Up

### Step 2: Verify Applications Are Accessible

# Test Flask Application (Port 5000):  
curl -s http://localhost:5000 | head -3  
  
# Test PWA Application (Port 9090):  
curl -s http://localhost:9090 | head -3

**Application Accessibility Check:**

* ☐ Flask App responds at http://localhost:5000 ✅
* ☐ PWA App responds at http://localhost:9090 ✅

### Step 3: Test DAST Tool

# Verify DAST analyzer is working:  
python src/analyzer/dast\_cli.py --help

**Tool Verification:**

* ☐ DAST CLI displays help information ✅

### Troubleshooting:

**If applications don’t respond:**

1. Restart containers: cd docker && docker-compose down && docker-compose up -d
2. Wait 30 seconds for applications to fully start
3. Re-test with curl commands above

**If you encounter any issues, notify your instructor before proceeding.**

## �📋 Exercise 1: DAST Fundamentals

### 1.1 Concept Understanding

**Complete the comparison table:**

| Testing Aspect | SAST (Static) | DAST (Dynamic) |
| --- | --- | --- |
| **Application must be running** | Yes / No | Yes / No |
| **Requires source code access** | Yes / No | Yes / No |
| **Tests actual user input** | Yes / No | Yes / No |
| **Finds runtime vulnerabilities** | Yes / No | Yes / No |
| **Speed of analysis** | Fast / Slow | Fast / Slow |

### 1.2 Tool Exploration

# Command run:  
python src/analyzer/dast\_cli.py --help

**List 3 scan modes available:**

**What file formats can reports be saved in?**

## 📋 Exercise 2: Basic Web Application Scanning

### 2.1 Flask Application Quick Scan

# Command run:  
python src/analyzer/dast\_cli.py http://localhost:5000 --quick --educational

**Scan Results Summary:**

* **Total Findings:** **\_\_\_\_**\_**\_\_\_\_**
* **Scan Duration:** **\_\_\_\_**\_**\_\_\_\_** seconds
* **HTTP Requests Made:** **\_\_\_\_**\_**\_\_\_\_**
* **Successful Responses:** **\_\_\_\_**\_**\_\_\_\_**

**Severity Breakdown:**

* **Critical:** **\_** **High:** **\_** **Medium:** **\_** **Low:** **\_** **Info:** **\_**

### 2.2 Security Headers Analysis

**List 3 missing security headers found:**

**Why are missing security headers a security concern?**

### 2.3 Deep Scan Analysis

# Command run:  
python src/analyzer/dast\_cli.py http://localhost:5000 --deep-scan --educational

**Additional findings from deep scan:**

* **Additional endpoints discovered:** **\_\_\_\_**
* **New vulnerabilities found:** **\_\_\_\_\_\*\*\_\_**\*\*
* **Total scan time difference:** **\_\_** seconds (quick vs deep)

**Which tools were used in the deep scan?**

## 📋 Exercise 3: PWA Application Analysis

### 3.1 Progressive Web App Scan

# Command run:  
python src/analyzer/dast\_cli.py http://localhost:9090 --educational --output pwa\_report.json

**PWA Scan Results:**

* **Total Findings:** **\_\_\_\_**\_**\_\_\_\_**
* **Most Severe Finding:** **\_\_\_\_**
* **Unique Vulnerabilities (not in Flask app):** **\_\_\_\_**\_**\_\_\_\_**

### 3.2 Application Comparison

**Complete the vulnerability comparison:**

| Vulnerability Type | Flask App | PWA App | Which is More Severe? |
| --- | --- | --- | --- |
| Missing Security Headers | Found / Not Found | Found / Not Found | Flask / PWA / Same |
| XSS Vulnerabilities | Found / Not Found | Found / Not Found | Flask / PWA / Same |
| SQL Injection | Found / Not Found | Found / Not Found | Flask / PWA / Same |
| Information Disclosure | Found / Not Found | Found / Not Found | Flask / PWA / Same |

**Which application has the higher overall risk score?**

## 📋 Exercise 4: Vulnerability Deep Dive

### 4.1 Cross-Site Scripting (XSS) Analysis

**If XSS was found, complete this section:**

**XSS Finding Details:**

* **Vulnerable Parameter:** **\_\_\_\_**
* **Test Payload Used:** **\_\_\_\_**\_\_\_\_\*\*
* **Evidence of Vulnerability:** **\_\_\_\_\_\*\*\_\_**\*\*

**How does the DAST scanner detect XSS?**

### 4.2 SQL Injection Analysis

**If SQL injection was found, complete this section:**

**SQL Injection Details:**

* **Vulnerable Parameter:** **\_\_\_\_**
* **Test Payload Used:** **\_\_\_\_**\_\_\_\_\*\*
* **Database Error Message:** **\_\_\_\_**\_\_\_\_\*\*

**Why do error messages indicate SQL injection vulnerability?**

### 4.3 Information Disclosure

**What sensitive information was disclosed by the applications?**

**How could an attacker use this information?**

## 📋 Exercise 5: SAST vs DAST Comparison

### 5.1 Combined Analysis Results

# Command run:  
python src/analyzer/dast\_cli.py --demo-apps --educational

**Compare with previous SAST results:**

| Vulnerability Category | SAST Found | DAST Found | Why the Difference? |
| --- | --- | --- | --- |
| **SQL Injection** | Yes / No | Yes / No | **\_\_\_\_**\_**\_\_\_\_** |
| **Cross-Site Scripting** | Yes / No | Yes / No | **\_\_\_\_**\_**\_\_\_\_** |
| **Missing Security Headers** | Yes / No | Yes / No | **\_\_\_\_**\_**\_\_\_\_** |
| **Debug Information** | Yes / No | Yes / No | **\_\_\_\_**\_**\_\_\_\_** |
| **Hardcoded Secrets** | Yes / No | Yes / No | **\_\_\_\_**\_**\_\_\_\_** |
| **Dependency Vulnerabilities** | Yes / No | Yes / No | **\_\_\_\_**\_**\_\_\_\_** |

### 5.2 Methodology Strengths

**List 2 advantages of DAST over SAST:**

**List 2 advantages of SAST over DAST:**

**How would you use both methods together in a security program?**

## 📋 Exercise 6: Professional Reporting

### 6.1 Executive Summary

**Write a brief executive summary of your findings:**

**DYNAMIC SECURITY ASSESSMENT SUMMARY**

**Applications Tested:** **\_\_\_\_\_\*\*\_\_**\*\*

**Total Security Issues Found:** **\_\_\_\_\_\*\*\_\_**\*\*

**Most Critical Finding:** **\_\_\_\_\_\*\*\_\_**\*\*

**Immediate Action Required:** **\_\_\_\_**\_\_\_\_\*\*

**Overall Risk Level:** Low / Medium / High / Critical

### 6.2 Top 3 Remediation Priorities

**1. Priority #1:**

* **Vulnerability:** **\_\_\_\_**\_\_\_\_\*\*
* **Risk Level:** **\_\_\_\_**\_**\_\*\*\_\_**\*\*
* **Remediation:** **\_\_\_\_**\_\_\_\_\*\*
* **Estimated Effort:** **\_\_\_\_**

**2. Priority #2:**

* **Vulnerability:** **\_\_\_\_**\_\_\_\_\*\*
* **Risk Level:** **\_\_\_\_**\_**\_\*\*\_\_**\*\*
* **Remediation:** **\_\_\_\_**\_\_\_\_\*\*
* **Estimated Effort:** **\_\_\_\_**

**3. Priority #3:**

* **Vulnerability:** **\_\_\_\_**\_\_\_\_\*\*
* **Risk Level:** **\_\_\_\_**\_**\_\*\*\_\_**\*\*
* **Remediation:** **\_\_\_\_**\_\_\_\_\*\*
* **Estimated Effort:** **\_\_\_\_**

### 6.3 Security Recommendations

**List 3 general security improvements for the applications:**

## 🎯 Reflection Questions

### Technical Understanding:

**1. What types of vulnerabilities can ONLY be found through dynamic testing?**

**2. Why is it important to test applications in a running state?**

**3. What are the limitations of DAST compared to SAST?**

### Practical Application:

**4. When would you run DAST scans in a development workflow?**

**5. How would you verify DAST findings before reporting them?**

### Career Relevance:

**6. What roles in cybersecurity would regularly use DAST tools?**

**7. How does DAST fit into compliance requirements (like ISM)?**

## 📚 Additional Learning

### Challenge Questions:

**1. Research: What is the difference between authenticated and unauthenticated DAST scanning?**

**2. Design: How would you integrate DAST into a CI/CD pipeline?**

**3. Analysis: What metrics would you track to measure DAST program effectiveness?**

## ⚖️ Legal and Ethical Considerations

### Professional Responsibility in Dynamic Testing

**1. Employment Impact:** How do runtime vulnerabilities you found affect developers and IT staff responsibilities?

**2. Privacy Rights:** What personal data could be exposed through the runtime vulnerabilities identified?

**3. Intellectual Property:** Could the security misconfigurations expose proprietary application logic?

### Regulatory Compliance

**4. Web Application Compliance:** How do missing security headers violate web security standards?

**5. Data Protection:** Which findings could lead to regulatory violations (Privacy Act,1988 (Privacy Act), ISM)?

### Ethical Testing Practices

**6. Authorized Testing:** Why is it critical to only perform DAST on applications you own or have permission to test?

**7. Responsible Disclosure:** How should runtime vulnerabilities be reported to application owners?

## 🔐 Cryptography and Runtime Security

### Cryptographic Implementation Assessment

**1. Transport Security:** Did you find issues with HTTPS implementation or weak encryption in transit?

**2. Session Management:** What cryptographic weaknesses were found in session handling?

**3. Authentication Security:** How do the authentication vulnerabilities relate to cryptographic best practices?

**4. Runtime Cryptography:** What recommendations would you make for improving cryptographic controls?

## 💼 Business Impact Assessment

### Enterprise Runtime Security Impact

**1. Operational Impact:** How would runtime exploitation of these vulnerabilities affect business operations?

**2. Customer Trust:** How could runtime security issues affect customer confidence and retention?

**3. Compliance Costs:** What would be the cost of regulatory fines from runtime security failures?

* **ISM Violations:** **\_\_\_\_**\_\_\_\_\*\*
* **Data Protection Fines:** **\_\_\_\_**\_**\_\*\*\_\_**\*\*
* **Industry-Specific Penalties:** **\_\_\_\_**

**4. Incident Response:** What would be the cost of responding to a security incident from these vulnerabilities?

**🎓 Completion Checklist:**

* ☐ Completed all scan commands successfully
* ☐ Analyzed findings from both applications
* ☐ Compared SAST vs DAST results
* ☐ Created professional remediation recommendations
* ☐ Reflected on practical applications of DAST

**Instructor Signature:** **\_\_\_\_** **Grade:** **\_\_\_**