Week 06: Security Mechanisms and Access Control

Learning Objectives

- Implement mandatory access control systems
- Configure intrusion detection tools
- Develop security verification scripts
- Create remote monitoring capabilities

Pre-Lab Preparation

Watch the following video lectures before attending the lab session:

- Mandatory Access Control Concepts
- Introduction to Intrusion Detection Systems
- Bash Scripting for System Administration
- Review: Operating System Modes

Lab and Coursework Activity: Security Hardening and Monitoring

Coursework Phase: Advanced Security and Monitoring Infrastructure

What you will do today:

- Implement AppArmor or SELinux mandatory access control
- Configure fail2ban for intrusion detection
- Set up automatic security updates
- Create a security baseline verification script
- Develop a remote monitoring script
- Document all security controls with evidence

Part 1: Mandatory Access Control Implementation

Task 1.1: Identifying Your Access Control System

Ubuntu typically uses AppArmor. Red Hat-based systems use SELinux.

Check which system is active:

```
# For AppArmor (Ubuntu/Debian)
sudo aa-status
```

```
# For SELinux (Red Hat/CentOS)
sestatus
```

For AppArmor Systems:

Task 1.2: Working with AppArmor Profiles

1. List all AppArmor profiles and their modes:

   ```
   sudo aa-status
   ```

2. Examine a specific profile:

   ```
   sudo cat /etc/apparmor.d/usr.sbin.tcpdump
   ```

3. Identify the permissions granted in this profile

4. View profiles in different modes:

   ```
   sudo aa-status --profiled
   sudo aa-status --enforced
   sudo aa-status --complaining
   ```

5. Understand the difference:

   - **Enforce mode**: Violations are blocked and logged
   - **Complain mode**: Violations are logged but allowed

For your journal: Document AppArmor status with screenshots. Explain the structure of an AppArmor profile and the purpose of enforce vs complain modes.

Task 1.3: Creating an AppArmor Status Report Script

1. Create a reporting script:

   ```
   nano apparmor-report.sh
   ```

2. Add the following content:

   ```
   #!/bin/bash
   # AppArmor Status Report Script
   # Reports on all AppArmor profiles and their status

   echo "========================================="
   echo "AppArmor Status Report"
   echo "========================================="
   echo "Generated: $(date)"
   echo "Hostname: $(hostname)"
   echo ""

   # Check if AppArmor is installed
   ```

```
if ! command -v aa-status &> /dev/null; then
    echo "ERROR: AppArmor is not installed"
    exit 1
fi

echo "=== Profile Summary ==="
# Count total profiles loaded
total_profiles=$(sudo aa-status --profiled | wc -l)
echo "Total profiles loaded: $total_profiles"
echo ""

echo "=== Enforced Profiles ==="
sudo aa-status --enforced
enforced_count=$(sudo aa-status --enforced | wc -l)
echo "Count: $enforced_count"
echo ""

echo "=== Complain Mode Profiles ==="
sudo aa-status --complaining
complain_count=$(sudo aa-status --complaining | wc -l)
echo "Count: $complain_count"
echo ""

echo "=========================================="
echo "Report Complete"
echo "=========================================="
```

3. Make it executable:

   ```
   chmod +x apparmor-report.sh
   ```

4. Run it:

   ```
   ./apparmor-report.sh
   ```

For SELinux Systems:

Task 1.2: Understanding SELinux Status

1. Check SELinux status:

   ```
   sestatus
   getenforce
   ```

2. View security contexts:

   ```
   ls -Z /etc/ssh/sshd_config
   ps -eZ | grep sshd
   ```

3. Check for recent denials:

```
sudo ausearch -m avc -ts recent
```

4. Create a similar reporting script adapted for SELinux

For your journal: Document your MAC system configuration with screenshots. Explain how mandatory access control differs from discretionary access control (traditional file permissions). Include your reporting script with line-by-line comments.

Part 2: Intrusion Detection with fail2ban

Task 2.1: Installing and Configuring fail2ban

1. Install fail2ban:

```
sudo apt update
sudo apt install fail2ban
```

2. Check service status:

```
sudo systemctl status fail2ban
```

3. Create a local configuration file (never edit jail.conf directly):

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

4. Edit the local configuration:

```
sudo nano /etc/fail2ban/jail.local
```

5. Locate the `[sshd]` section and configure:

```
[sshd]
enabled = true
port = 22
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
bantime = 600
findtime = 600
```

Parameters explained:

– **maxretry**: Number of failed attempts before ban

– **bantime**: How long to ban (seconds)

– **findtime**: Time window for counting failures

6. Save and restart fail2ban:

```
sudo systemctl restart fail2ban
sudo systemctl enable fail2ban
```

For your journal: Screenshot your fail2ban configuration. Explain each parameter and justify your choices.

Task 2.2: Monitoring fail2ban Activity

1. Check active jails:

   ```
   sudo fail2ban-client status
   sudo fail2ban-client status sshd
   ```

2. View fail2ban log:

   ```
   sudo tail -30 /var/log/fail2ban.log
   ```

3. To test (optional, requires second system or terminal):

   - Attempt multiple failed SSH logins with wrong password
   - Check if IP gets banned:

     ```
     sudo fail2ban-client status sshd
     ```

4. View banned IPs:

   ```
   sudo fail2ban-client banned
   ```

5. Manually unban an IP if needed:

   ```
   sudo fail2ban-client set sshd unbanip [IP_ADDRESS]
   ```

6. Examine authentication logs for failed attempts:

   ```
   sudo grep "Failed password" /var/log/auth.log | tail -10
   ```

For your journal: Document fail2ban status and activity logs. Explain how fail2ban protects against brute-force attacks and the relationship between maxretry, bantime, and findtime.

Task 2.3: Configuring Automatic Security Updates

1. Install unattended-upgrades:

   ```
   sudo apt install unattended-upgrades
   ```

2. Enable automatic security updates:

   ```
   sudo dpkg-reconfigure -plow unattended-upgrades
   ```

   Select "Yes" when prompted

3. Verify the service is running:

   ```
   sudo systemctl status unattended-upgrades
   ```

4. Check the configuration:

```
sudo cat /etc/apt/apt.conf.d/50unattended-upgrades
```

5. View update logs:

```
sudo cat /var/log/unattended-upgrades/unattended-upgrades.log
```

For your journal: Document automatic updates configuration. Discuss the security vs stability trade-off of automatic updates.

Part 3: Security Verification and Monitoring Scripts

Task 3.1: Creating the Security Baseline Verification Script

1. Create the script file:

```
nano security-baseline.sh
```

2. Add comprehensive security checks:

```
#!/bin/bash
# Security Baseline Verification Script
# Verifies all security configurations from Phase 05 and Phase 06
# This script runs on the server (executed via SSH)

echo "=========================================="
echo "Security Baseline Verification Report"
echo "=========================================="
echo "Generated: $(date)"
echo "Hostname: $(hostname)"
echo "Server IP: $(hostname -I | awk '{print $1}')"
echo ""

# Colour codes for output
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Colour

# Check SSH configuration
echo "=== SSH Security Configuration ==="

# Check password authentication
echo -n "Password Authentication: "
if grep -q "^PasswordAuthentication no" /etc/ssh/sshd_config; then
    echo -e "${GREEN}DISABLED${NC} (Secure)"
else
    echo -e "${RED}ENABLED${NC} (Warning: Should be disabled)"
fi
```

```
# Check root login
echo -n "Root Login via SSH: "
if grep -q "^PermitRootLogin no" /etc/ssh/sshd_config; then
    echo -e "${GREEN}DISABLED${NC} (Secure)"
else
    echo -e "${RED}ENABLED${NC} (Warning: Should be disabled)"
fi

# Check public key authentication
echo -n "Public Key Authentication: "
if grep -q "^PubkeyAuthentication yes" /etc/ssh/sshd_config; then
    echo -e "${GREEN}ENABLED${NC} (Secure)"
else
    echo -e "${YELLOW}DISABLED${NC} (Warning: Should be enabled)"
fi

echo ""

# Check firewall status
echo "=== Firewall Configuration ==="
if command -v ufw &> /dev/null; then
    echo "Firewall Status:"
    sudo ufw status
    echo ""
    echo "Active Rules:"
    sudo ufw status numbered
else
    echo -e "${RED}UFW not installed${NC}"
fi
echo ""

# Check fail2ban status
echo "=== Intrusion Detection (fail2ban) ==="
if systemctl is-active --quiet fail2ban; then
    echo -e "Service Status: ${GREEN}RUNNING${NC} (Secure)"
    echo ""
    echo "Active Jails:"
    sudo fail2ban-client status
    echo ""
    echo "SSH Jail Status:"
    sudo fail2ban-client status sshd 2>/dev/null || echo "SSH jail not
configured"
else
    echo -e "Service Status: ${RED}NOT RUNNING${NC} (Warning)"
fi
echo ""

# Check mandatory access control
echo "=== Mandatory Access Control ==="
```

```bash
if command -v aa-status &> /dev/null; then
    echo "System: AppArmor"
    echo "Status:"
    sudo aa-status --profiled 2>/dev/null | head -5
    echo ""
    enforced=$(sudo aa-status --enforced 2>/dev/null | wc -l)
    echo "Profiles in enforce mode: $enforced"
elif command -v sestatus &> /dev/null; then
    echo "System: SELinux"
    sestatus | grep "SELinux status"
    sestatus | grep "Current mode"
else
    echo -e "${RED}No MAC system detected${NC} (Warning)"
fi
echo ""

# Check administrative users
echo "=== Administrative Users ==="
echo "Users with sudo privileges:"
getent group sudo | cut -d: -f4
echo ""

# Check automatic updates
echo "=== Automatic Security Updates ==="
if dpkg -l | grep -q unattended-upgrades; then
    echo -e "unattended-upgrades: ${GREEN}INSTALLED${NC}"
    if systemctl is-enabled unattended-upgrades &> /dev/null; then
        echo -e "Status: ${GREEN}ENABLED${NC} (Secure)"
    else
        echo -e "Status: ${YELLOW}DISABLED${NC} (Warning)"
    fi
else
    echo -e "unattended-upgrades: ${RED}NOT INSTALLED${NC} (Warning)"
fi
echo ""

# System resource check
echo "=== System Resources ==="
echo "Memory Usage:"
free -h | grep -E "Mem|Swap"
echo ""
echo "Disk Usage:"
df -h | grep -E "^/dev"
echo ""

echo "=========================================="
echo "Security Baseline Verification Complete"
echo "=========================================="
```

3. Make it executable:

```
chmod +x security-baseline.sh
```

4. Test the script:

```
./security-baseline.sh
```

5. Review the output and address any warnings

For your journal: Include the complete script with line-by-line comments explaining each check. Screenshot showing the script output with all security controls verified.

Task 3.2: Creating the Remote Monitoring Script

1. On your workstation, create a remote monitoring script:

```
nano monitor-server.sh
```

2. Add comprehensive monitoring capabilities:

```
#!/bin/bash
# Remote Server Monitoring Script
# Runs on workstation, connects to server via SSH
# Collects performance and security metrics

# Configuration - UPDATE THESE VALUES
SERVER="username@server_ip"
LOGFILE="monitoring-$(date +%Y%m%d_%H%M%S).log"

# Colour codes
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

echo "==========================================="
echo "Remote Server Monitoring Report"
echo "==========================================="
echo "Generated: $(date)"
echo "Monitoring: $SERVER"
echo "Log file: $LOGFILE"
echo ""

# Function to collect and display metrics
collect_metrics() {
    echo "=== System Information ===" | tee -a $LOGFILE
    ssh $SERVER 'uname -a' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== Uptime and Load ===" | tee -a $LOGFILE
    ssh $SERVER 'uptime' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE
    echo "=== CPU Usage (top 10 processes) ===" | tee -a $LOGFILE
    ssh $SERVER 'ps aux --sort=-%cpu | head -11' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== Memory Usage ===" | tee -a $LOGFILE
    ssh $SERVER 'free -h' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== Disk Usage ===" | tee -a $LOGFILE
    ssh $SERVER 'df -h' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== Disk I/O Statistics ===" | tee -a $LOGFILE
    ssh $SERVER 'iostat -x 1 2 | tail -10' 2>/dev/null | tee -a
$LOGFILE
    if [ $? -ne 0 ]; then
        echo "iostat not available (install sysstat package)" | tee -a
$LOGFILE
    fi
    echo "" | tee -a $LOGFILE

    echo "=== Network Connections ===" | tee -a $LOGFILE
    ssh $SERVER 'ss -s' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== Active Network Connections (top 10) ===" | tee -a
$LOGFILE
    ssh $SERVER 'ss -tupn | head -11' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== Recent Failed Login Attempts ===" | tee -a $LOGFILE
    ssh $SERVER 'sudo grep "Failed password" /var/log/auth.log
2>/dev/null | tail -5' | tee -a $LOGFILE
    echo "" | tee -a $LOGFILE

    echo "=== fail2ban Status ===" | tee -a $LOGFILE
    ssh $SERVER 'sudo fail2ban-client status 2>/dev/null' | tee -a
$LOGFILE
    if [ $? -ne 0 ]; then
        echo "fail2ban not available" | tee -a $LOGFILE
    fi
    echo "" | tee -a $LOGFILE
}

# Check if SSH connection works
echo -n "Testing SSH connection... "
if ssh -o ConnectTimeout=5 $SERVER 'exit' 2>/dev/null; then
    echo -e "${GREEN}SUCCESS${NC}"
```

```
        echo ""
        collect_metrics
    else
        echo -e "${YELLOW}FAILED${NC}"
        echo "Cannot connect to $SERVER"
        echo "Please check:"
        echo "1. Server is running"
        echo "2. SSH service is active"
        echo "3. Firewall allows connection from this IP"
        echo "4. SSH key authentication is configured"
        exit 1
    fi

    echo "==========================================="
    echo "Monitoring Complete"
    echo "==========================================="
    echo "Full log saved to: $LOGFILE"
```

3. Make it executable:

   ```
   chmod +x monitor-server.sh
   ```

4. Test the script:

   ```
   ./monitor-server.sh
   ```

5. Review the monitoring data collected

For your journal: Include the complete monitoring script with explanations of each metric collected. Screenshots showing the script running from your workstation and successfully collecting server metrics remotely.

Journal Entry Requirements for This Week

Your journal entry must include:

Mandatory Access Control Section:

- AppArmor or SELinux status with screenshots
- Explanation of profiles/contexts and their modes
- AppArmor/SELinux reporting script with line-by-line comments
- Discussion of how MAC differs from DAC (discretionary access control)

Intrusion Detection Section:

- fail2ban installation and configuration documentation

- Screenshots of fail2ban status and active jails
- Explanation of configuration parameters (maxretry, bantime, findtime)
- Evidence of fail2ban protecting against brute-force attempts

Automatic Updates Section:

- Configuration evidence with screenshots
- Discussion of security vs stability trade-off
- Justification for your update strategy

Security Baseline Script:

- Complete script with comprehensive line-by-line comments
- Screenshots showing script execution and output
- Explanation of each security check performed
- Any warnings identified and how you addressed them

Remote Monitoring Script:

- Complete script with line-by-line comments
- Screenshots showing remote execution from workstation
- Explanation of metrics collected and their significance
- Evidence that monitoring works via SSH without accessing server console

Reflection:

- Challenges in implementing security controls
- Trade-offs between security, performance, and usability
- How Phase 06 builds upon Phase 05 security foundation
- Connection to learning outcomes (LO3, LO4)

Technical Requirements:

- All screenshots must show username@hostname
- Scripts must include error handling and informative output
- Update architecture diagram to show all security layers
- Commit to GitHub: `git add . && git commit -m "Week 06: Advanced security implementation" && git push`