Preventing Vulnerabilities in DevOps Scripts

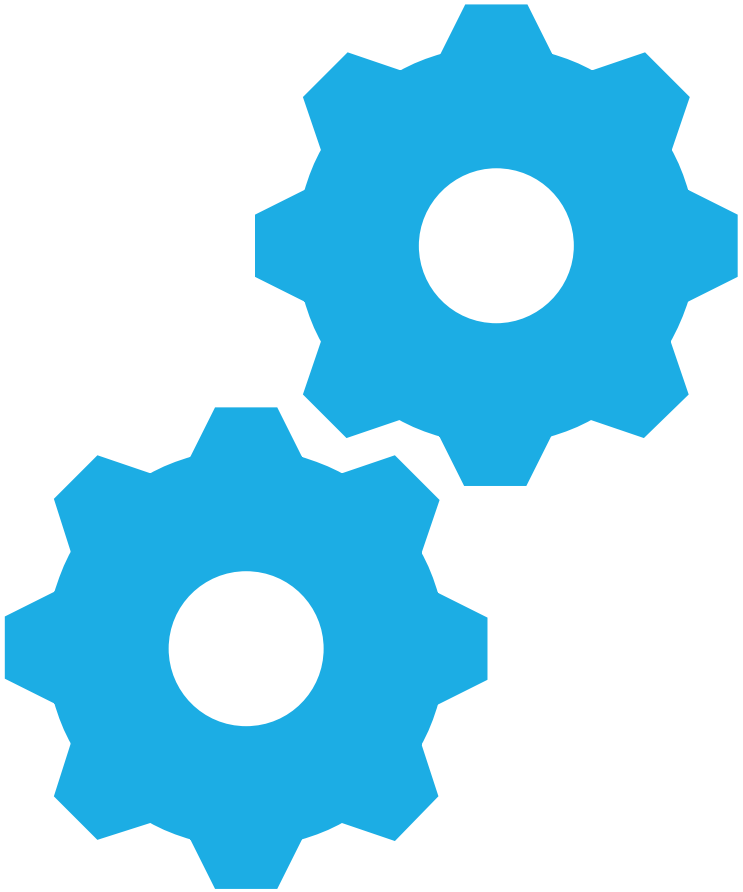
PALLAVI VAIDYA

Agenda

What DevOps scripts are and their role in automation.

Common vulnerabilities in DevOps scripts.

Techniques and tools to secure them.



DevOps script

DevOps scripts are automated sets of instructions used to perform repetitive or complex tasks in software development and IT operations.

They help ensure consistency, speed, and reliability in deploying, configuring, and managing systems or applications.

DevOps scripts are programs or command sequences written in languages such as Bash, PowerShell, Python, or Groovy that automate key stages of the DevOps lifecycle—from development to deployment and monitoring.

Purpose

Automate manual tasks (e.g., starting servers, deploying apps, setting environment variables)

Reduce human error in repetitive processes

Ensure consistency across environments (development, testing, production)

Speed up CI/CD pipelines (Continuous Integration / Continuous Deployment)

Why It Matters??

Because DevOps scripts often handle credentials, environment variables, and deployments, insecure scripting can lead to:

- Unauthorized access
- Data leaks
- Privilege escalation
- Service downtime

Common Vulnerabilities in DevOps Scripts

- 🔑 Hardcoded Secrets
- 💣 Command Injection
- 🔐 Excessive Privileges
- 📂 Unsafe File Handling
- 🌐 Unverified Downloads or Dependencies
- 🪵 Sensitive Data in Logs
- ⚙️ Environment Misconfigurations
- 🧱 Lack of Error Handling

Hardcoded Secrets

Problem: Passwords, API keys, or tokens stored directly in scripts.

Risk: If the script is shared or exposed (e.g., on GitHub), attackers can gain system access.

Fix: Use secret management tools like Vault, AWS Secrets Manager, or environment variables.

Command Injection

Problem: Accepting unvalidated user input that executes system commands.

Risk: Attackers can run malicious commands on the server.

Fix: Validate inputs or use safe libraries (subprocess.run() with argument lists).

Excessive Privileges

Problem: Scripts running as root or with full admin access unnecessarily.

Risk: Increases potential damage if compromised.

Fix: Follow the Principle of Least Privilege — grant only required permissions.

Unsafe File Handling

Problem: Scripts read/write temporary or configuration files without checking permissions.

Risk: Sensitive data can be leaked or modified.

Fix: Set secure file permissions and avoid writing secrets to disk.

Unverified Downloads or Dependencies

Problem: Scripts install packages or binaries without verifying authenticity.

Risk: Attackers can inject malware through tampered sources.

Fix: Verify checksums, use signed packages, and trusted repositories.

Sensitive Data in Logs

Problem: Logging credentials, tokens, or personal data.

Risk: Logs may be accessible to unauthorized users.

Fix: Mask or omit sensitive data in logs.

Environment Misconfigurations

Problem: Scripts misconfigure servers or containers (e.g., exposing open ports).

Risk: Attackers can exploit misconfigurations to gain access.

Fix: Use configuration scanners (like Lynis, Trivy, or Scout Suite) to detect issues.

Lack of Error Handling

Problem: Scripts that fail silently or ignore errors.

Risk: Undetected failures can leave systems in insecure states.

Fix: Add proper error handling and exit codes.

Tools for Securing DevOps Scripts

- 🖌️ Static Code Analysis Tools (Bandit)
- 🔍 Dependency and Package Scanners(Audit)
- 🧪 Secrets Scanning with GitLeaks
- 🧩 ShellCheck for Shell Scripts

Best Practices

Manage secrets securely

Validate and sanitize inputs

Follow the principle of least privilege

Secure file and log handling

Verify external dependencies

Automate security checks

Implement secure configuration management

Conduct regular audits and code reviews

Monitor and respond

Validate and Sanitize Inputs

Trust	- Never trust user input — always validate and escape it
Prevent	- Prevent command injection and path traversal attacks
Use	- Use safe API functions (subprocess.run([...]) instead of os.system())
Validate	- Validate parameters before passing to shell commands

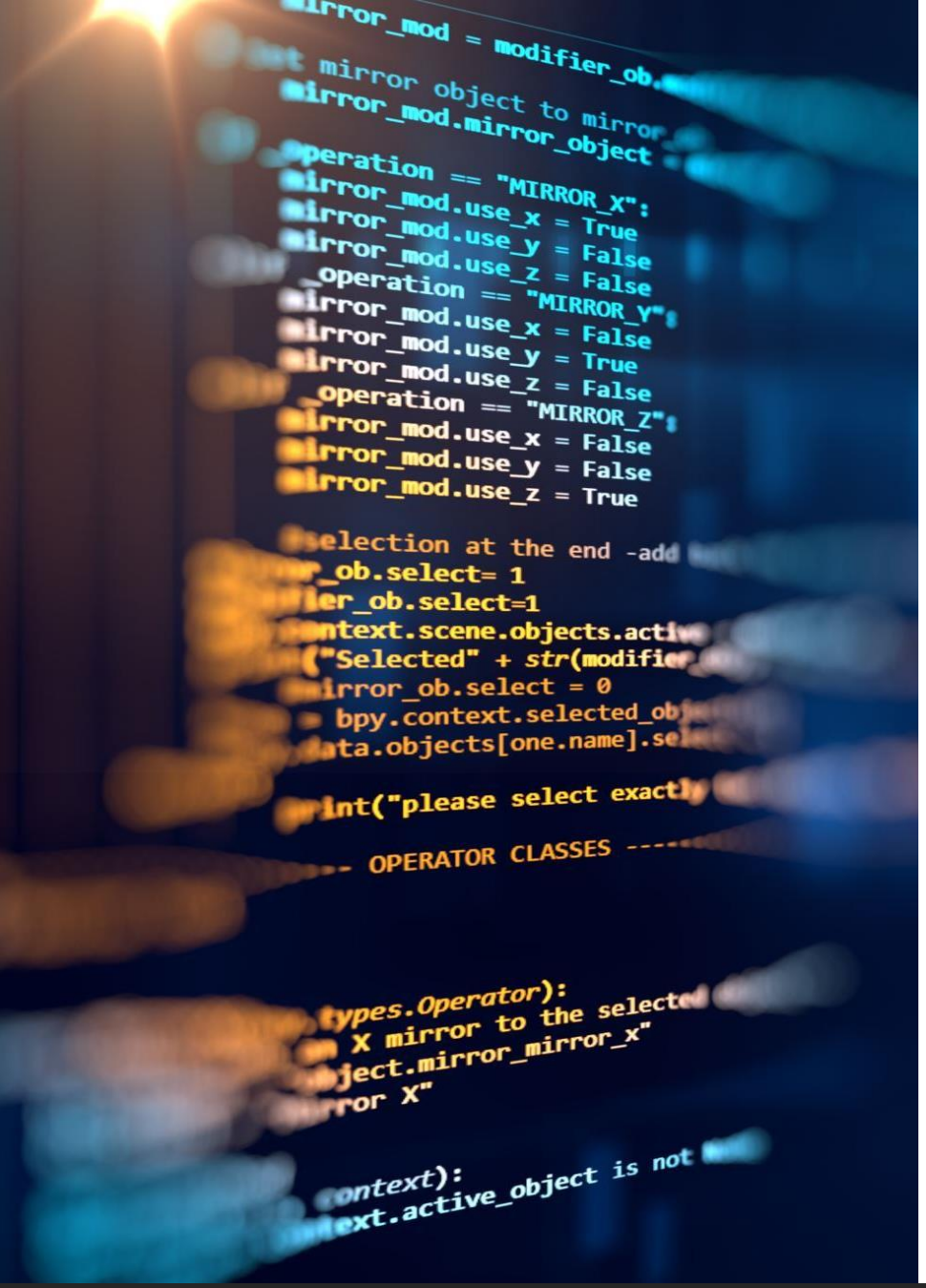
Follow the Principle of Least Privilege

- Run scripts with minimum required permissions

- Avoid using root or sudo unless necessary

- Separate administrative and non-administrative tasks

- Audit permission usage regularly



Secure File and Log Handling

- Avoid storing sensitive data in plain text

- Set file permissions (chmod 600) for configs

- Sanitize logs — don’t include credentials or tokens

- Encrypt sensitive files at rest and in transit

Verify External Dependencies

- Only download from trusted sources

- Use checksums and digital signatures

- Regularly update dependencies (patch vulnerabilities)

- Use scanners like Snyk or Trivy

Automate Security Checks

- Integrate tools in CI/CD pipelines:
  * Bandit (Python), ShellCheck (Bash)
  * SonarQube, GitHub Advanced Security
- Run security scans before deployments
- Fail builds if vulnerabilities are detected

Implement Secure Configuration Management

- Use IaC tools (Terraform, Ansible) securely

- Store configuration separately from code

- Validate configuration syntax and permissions

- Scan infrastructure with Checkov or Scout Suite

Conduct Regular Audits and Code Reviews

Peer	- Peer review all automation scripts
Review	- Review logs and CI/CD audit trails
Use	- Use version control to track changes
Perform	- Perform regular penetration tests on automation pipelines

Monitor and Respond

Implement	- Implement runtime monitoring (Falco, Wazuh)
Set	- Set alerts for unusual script activity
Maintain	- Maintain incident response procedures
Log	- Log all script executions for traceability