Break things, but not security:
CI/CD done right

Gijs Van Laer

SecAppDev

About me

CTO at XFA, a cybersecurity company focused on device security

Previous

- Information Security Consultant
- CISO at DPG Media
- Software developer

Education

- PhD in Cryptography (advised by Matt Green), Johns Hopkins University
- Master of Science in Security Informatics, Johns Hopkins University
- Master of Science in Pure Mathematics, University of Antwerp

gijs.vanlaer@xfa.tech

SecAppDev

The audience (by show of hands)

- Who owns or maintains a CI/CD pipeline?
- For developers in the room:
  - Do you get code in Production without approval of another person?
  - How fast is your code in production after approval?
    - More than an hour?
    - Between 30 minutes and 1 hour?
    - Between 10 minutes and 30 minutes?
    - Between 5 and 10 minutes?
    - Below 5 minutes?
- Does anyone in the room have to triage findings of a SAST or DAST?
- Who has ever bypassed a security check to “get things done”?

SecAppDev

Why Secure CI/CD?

- CI/CD is the backbone of modern software delivery
- Pipelines now execute code, manage secrets, deploy to prod
- They've become high-value targets

SecAppDev

OWASP TOP 10
CI/CD SECURITY RISKS
(2023)

10. Insufficient Logging & Visibility
9. Improper Artifact Integrity Validation
8. Ungoverned Usage of 3rd Party Services
7. Insecure System Configuration
6. Insufficient Credential Hygiene
5. Insufficient PBAC (Pipeline-Based Access Controls)
4. Poisoned Pipeline Execution (PPE)
3. Dependency Chain Abuse
2. Inadequate Identity and Access Management
1. Insufficient Flow Control Mechanisms

SecAppDev

10. Insufficient Logging & Visibility - Risk Explained

- Lack of logs for key pipeline events (e.g. job runs, artifact access)
- No visibility into who triggered what, when, and how
- Missed detection of anomalies or breaches

SecAppDev

10. Insufficient Logging & Visibility - Best practices

- Mapping the environment
- Enable full audit logging for pipeline runs, secrets access, approvals
- Centralize logs securely (e.g. SIEM)
- Use alerting for abnormal access patterns
- Redact secrets in logs

SecAppDev

9. Improper Artifact Integrity Validation - Real World Case: Codecov Bash Uploader Breach (2021)

Summary:

- Attackers gained unauthorized access to Codecov's Google Cloud Storage bucket, which hosted the Bash Uploader script.
- They modified the script to exfiltrate environment variables, including sensitive credentials, from users' CI environments.
- The malicious script was distributed to users for over two months before detection.

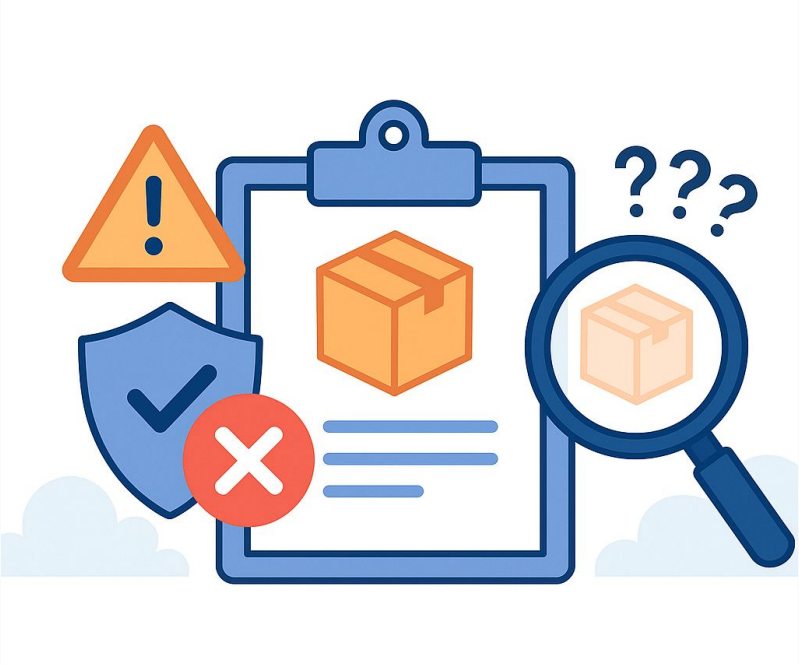
Impact:

- Secrets from thousands of CI/CD pipelines were compromised.
- The breach affected numerous organizations, leading to widespread credential rotations and security audits.

Source: https://about.codecov.io/security-update/

SecAppDev

9. Improper Artifact Integrity Validation - Risk Explained



- Artifacts (build outputs) not verified before promotion/deploy
- Tampered or stale artifacts may be trusted as valid
- Lack of inventory = unclear origin, ownership, or integrity

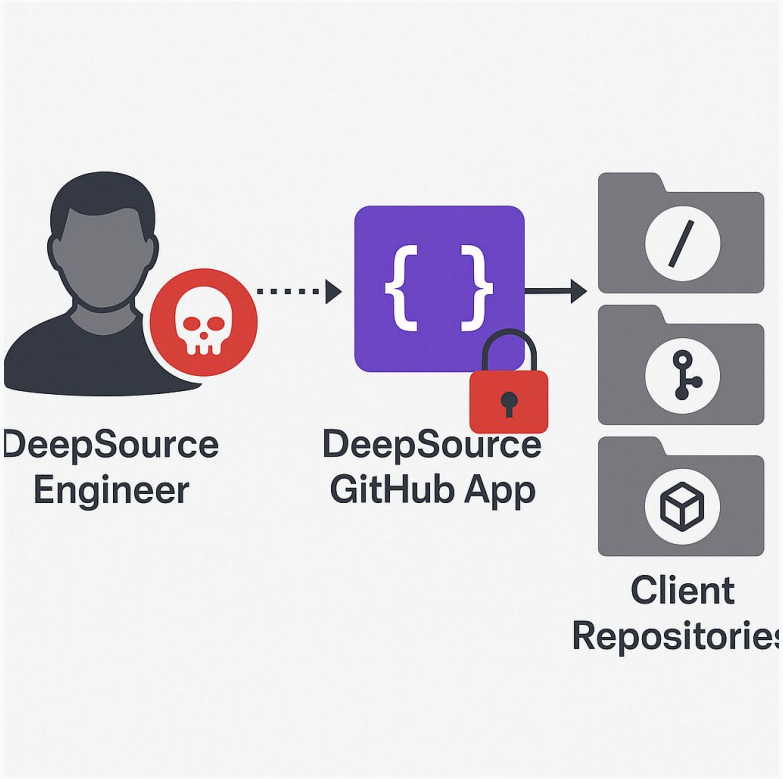
SecAppDev

9. Improper Artifact Integrity Validation - Best practices

- Enforce checksums/signatures before promotion
- Use immutable, versioned artifacts
- Store artifacts in controlled, access-logged registries
- Implement provenance metadata and SBOMs
- Alert on untracked or unverified artifacts

SecAppDev

8. Ungoverned Usage of 3rd Party Services - Real World Case: DeepSource GitHub App Compromise (2020)



Summary:

- An attacker compromised a DeepSource engineer's GitHub account via a phishing campaign.
- This granted the attacker access to the DeepSource GitHub App's credentials.
- Using these credentials, the attacker accessed client repositories of organizations that had installed the DeepSource GitHub App.

Impact:

- Unauthorized access to multiple client codebases.
- Potential exposure of sensitive code and data across various organizations.

Source: https://discuss.deepsource.com/t/security-incident-on-deepsource-s-github-application/131

SecAppDev

8. Ungoverned Usage of 3rd Party Services - Risk Explained

- Overuse of SaaS tools, integrations, bots in CI/CD
- No review or control over what they access or do
- Weak link in the pipeline — external but deeply embedded

SecAppDev

8. Ungoverned Usage of 3rd Party Services
- Best practices

- Vet all 3rd-party services before integration
- Use least privilege for app tokens and permissions
- Review app scopes regularly and revoke unused ones
- Prefer self-hosted, auditable alternatives for sensitive steps
- Monitor and alert on 3rd-party app activity

SecAppDev

7. Insecure System Configuration - Real World Case: SolarWinds Orion Build System Compromise (2020)

Summary:

- In 2020, attackers infiltrated SolarWinds' CI/CD pipeline by exploiting insecure system configurations.
- They introduced a malicious backdoor, known as SUNBURST, into the Orion software during the build process.
- The compromised software was digitally signed and distributed to approximately 18,000 customers, including U.S. government agencies and Fortune 500 companies. OBJ OBJ

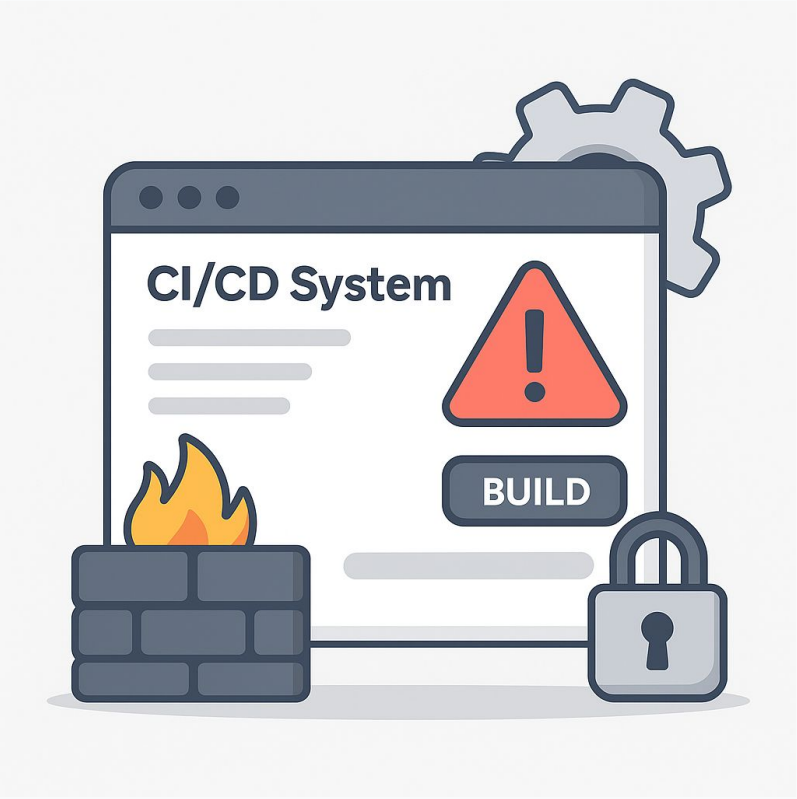
Impact:

- Widespread deployment of malicious software across numerous organizations.
- Extended dwell time allowed attackers to conduct espionage and data exfiltration.
- Significant reputational and financial damage to SolarWinds and affected entities.

Source: https://www.lawfaremedia.org/article/solarwinds-and-holiday-bear-campaign-case-study-classroom

SecAppDev

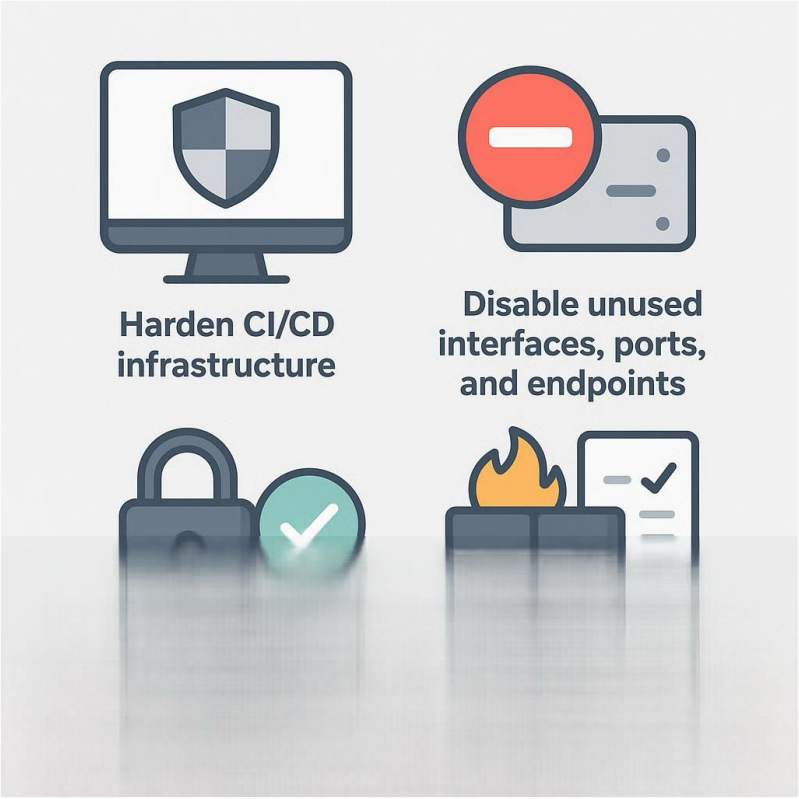
7. Insecure System Configuration - Risk Explained



- CI/CD systems exposed to public networks
- Default credentials, outdated software, poor hardening
- Overprivileged runners or shared agents

SecAppDev

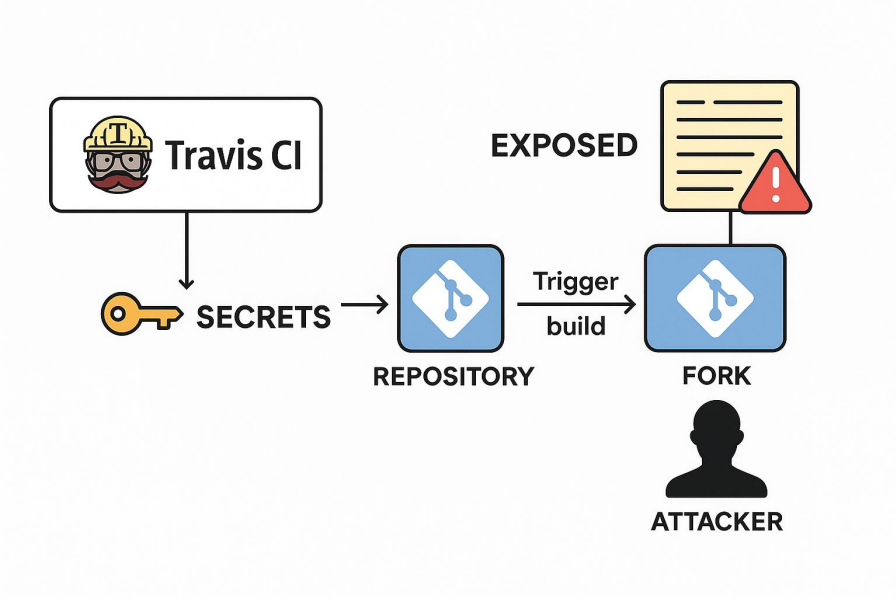
7. Insecure System Configuration - Best practices



- Harden CI/CD infrastructure (OS, runtime, containers)
- Disable unused interfaces, ports, and endpoints
- Enforce strong authentication and role-based access
- Update dependencies and plugins regularly
- Restrict network exposure with allowlists and firewalls

SecAppDev

6. Insufficient Credential Hygiene - Real World Case: Travis CI Secrets Exposure (2021)



Summary:

- A vulnerability in Travis CI exposed sensitive environment variables, including API keys and tokens.
- The flaw occurred when Travis CI improperly shared these variables with builds from forked repositories.
- Attackers could exploit this by creating malicious pull requests to trigger builds and then querying the Travis CI API to extract these secrets.

Impact:

- Many affected projects and organizations had to rotate their exposed credentials and conduct security audits to prevent unauthorized access.
- The incident underscored the risks associated with relying on cloud-based CI/CD services without proper security controls and secret management policies.

Source: https://travis-ci.community/t/security-bulletin/12081

SecAppDev

6. Insufficient Credential Hygiene - Risk Explained

- Secrets hardcoded in scripts, repos, or env files
- Long-lived tokens with broad scope and no rotation
- Exposure via logs, forks, or public workflows

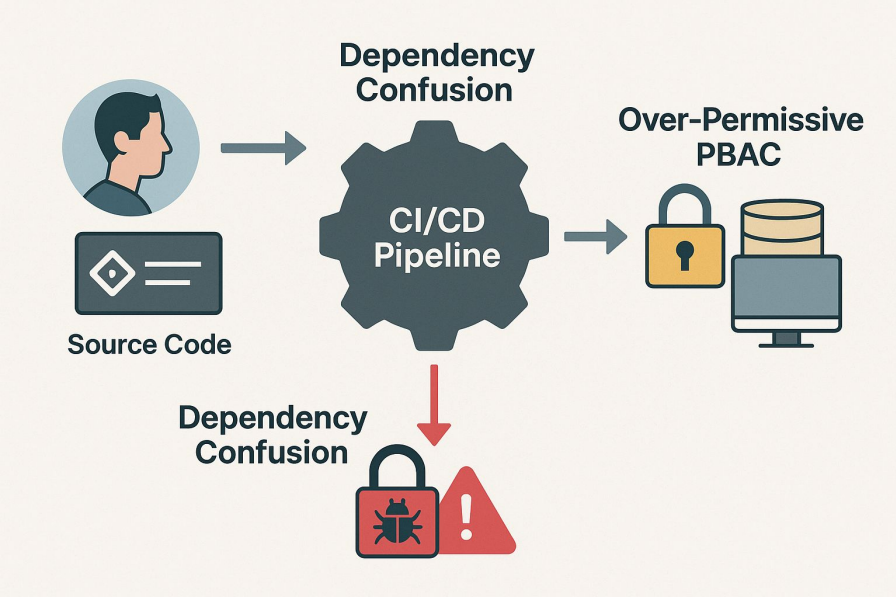
SecAppDev

6. Insufficient Credential Hygiene - Best practices

- Store credentials in secret management systems (Vault, AWS Secrets Manager, etc.)
- Avoid storing secrets in repo history or config files
- Use short-lived, scoped tokens with rotation
- Never expose secrets via logs or outputs
- Automate detection with tools (e.g. TruffleHog, Gitleaks)

SecAppDev

5. Insufficient PBAC (Pipeline-Based Access Controls)- Real World Case: Dependency Confusion Leading to PBAC Abuse (2021)



Summary:

- Attackers exploited dependency confusion vulnerabilities in Node.js applications of companies like Amazon, Zillow, Lyft, and Slack.
- By publishing malicious packages with names matching internal dependencies to public registries.
- The malicious code executed within the pipeline's context, which had excessive permissions due to insufficient PBAC.

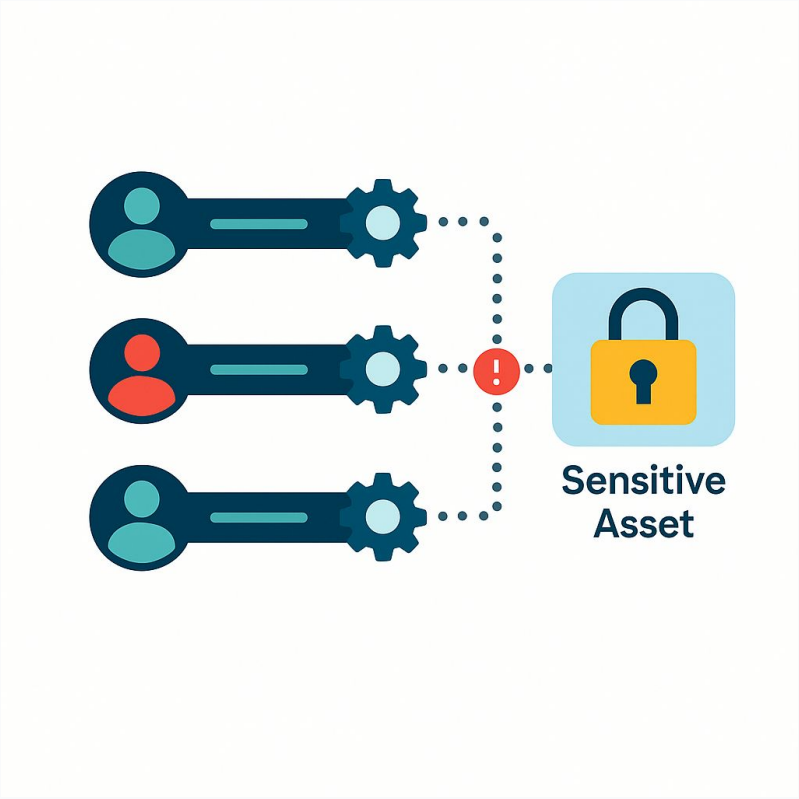
Impact:

- Execution of malicious code within CI/CD pipelines.
- Access to sensitive data and systems beyond the intended scope.
- Potential lateral movement within the organization's infrastructure.

Source: https://www.bleepingcomputer.com/news/security/malicious-npm-packages-target-amazon-slack-with-new-dependency-attacks/

SecAppDev

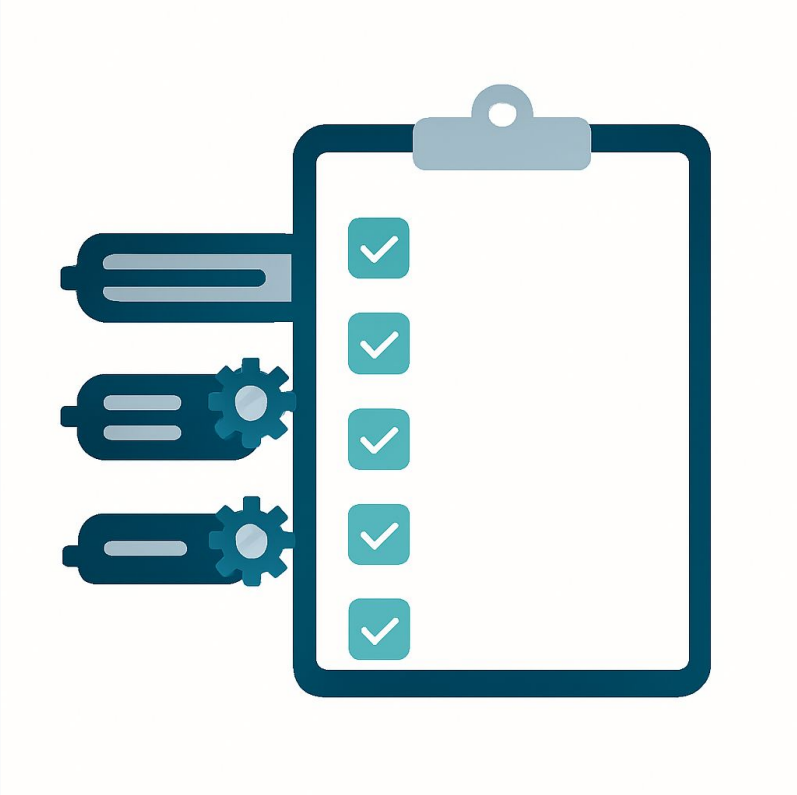
5. Insufficient PBAC (Pipeline-Based Access Controls)- Risk Explained



- All pipelines or jobs can access shared credentials/artifacts
- Developer pipelines run with excessive privileges
- Lack of isolation allows lateral movement between jobs

SecAppDev

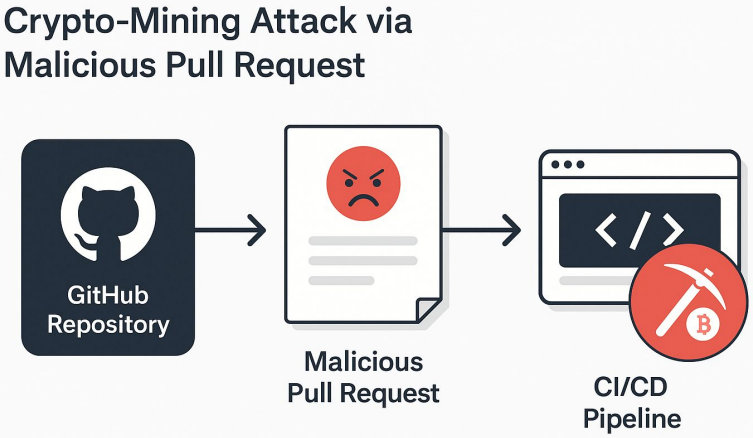
5. Insufficient PBAC (Pipeline-Based Access Controls)- Best practices



- Define fine-grained access rules per job, repo, and pipeline
- Isolate secrets per stage and environment
- Avoid sharing credentials or tokens across pipelines
- Use identity-aware execution (e.g., OIDC-based per-job auth)
- Audit pipeline permissions regularly

SecAppDev

4. Poisoned Pipeline Execution (PPE) - Real World Case: Direct PPE via GitHub Actions (2021)



Summary:

- A GitHub user submitted a pull request to a repository, introducing a malicious GitHub Actions workflow file named ci.yml.
- The workflow was configured to trigger on pull_request events and included a base64-encoded command that, when decoded, initiated a cryptocurrency mining operation.
- The attacker repeatedly opened and closed the pull request to trigger multiple workflow runs.

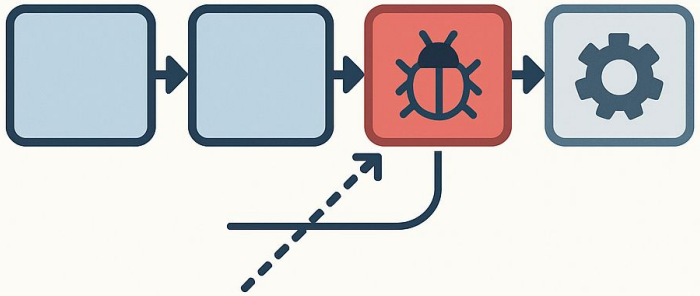
Impact:

- Unauthorized consumption of GitHub's infrastructure resources for illicit cryptocurrency mining.
- Potential for similar attacks to execute arbitrary code, leading to data exfiltration or further compromise of CI/CD pipelines.

Source: https://dev.to/thibaultduponchelle/the-github-action-mining-attack-through-pull-request-2lmc

SecAppDev

4. Poisoned Pipeline Execution (PPE) - Risk Explained



- Untrusted code runs in trusted CI/CD context
- Malicious steps injected via PRs, branches, jobs, or dependencies
- Attackers “poison” the build process for persistence or lateral movement

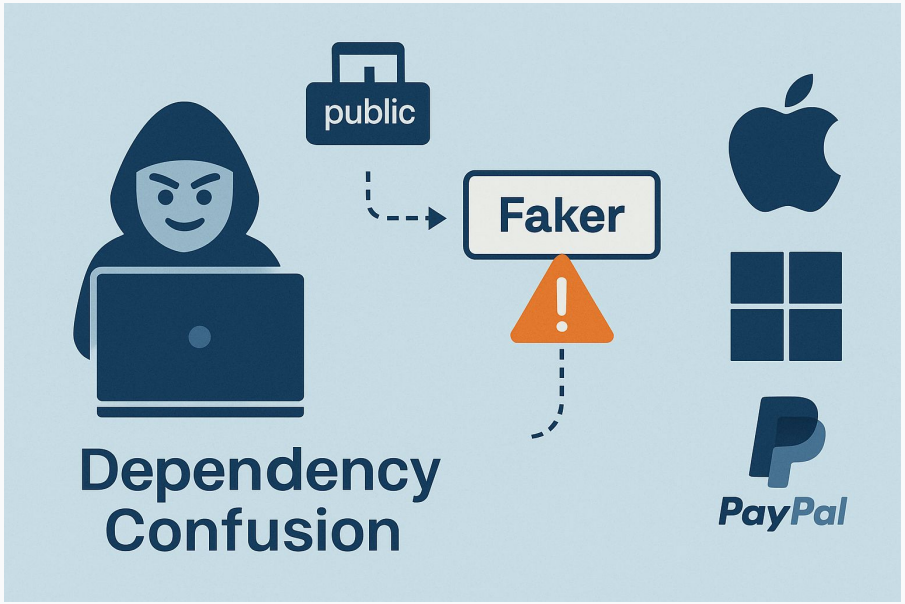
SecAppDev

4. Poisoned Pipeline Execution (PPE) - Best practices

- Review and restrict pipeline trigger conditions (e.g., only on trusted branches)
- Run untrusted code in isolated, sandboxed runners
- Validate workflow files and build scripts via policy-as-code
- Use ephemeral environments for PR builds
- Scan and verify any fetched dependencies or artifacts

SecAppDev

3. Dependency Chain Abuse - Real World Case: Dependency Confusion Attack on Major Tech Companies (2021)



Summary:

- Security researcher Alex Birsan identified that many organizations use internal packages not present in public repositories.
- By uploading malicious packages with the same names to public registries like npm, PyPI, and RubyGems, he exploited package managers' default behavior to prioritize public packages with higher version numbers.
- This led to the inadvertent installation and execution of his code within the internal systems of over 35 major companies, including Apple, Microsoft, and PayPal.

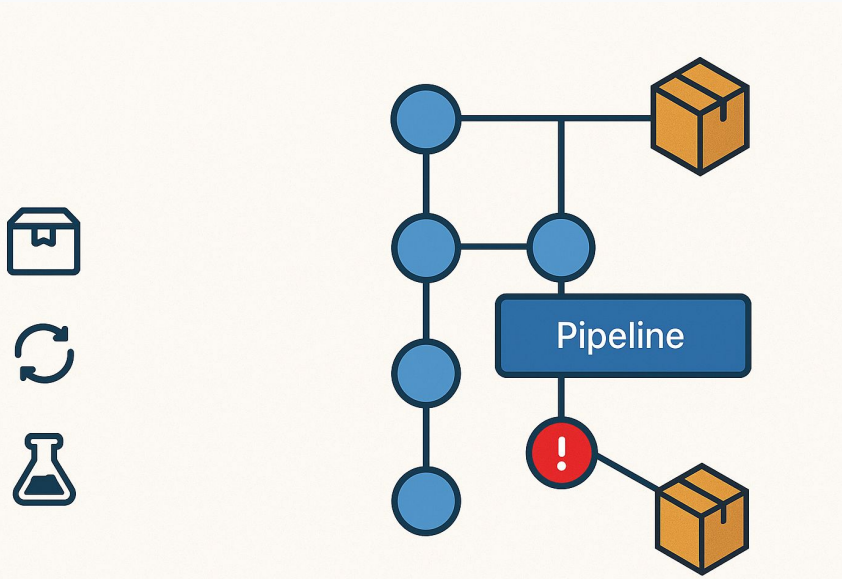
Impact:

- Unauthorized code execution within corporate networks.
- Potential exposure of sensitive data and internal systems.
- Highlighting systemic vulnerabilities in software supply chains.

Source: https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

SecAppDev

3. Dependency Chain Abuse - Risk Explained



- Trusting public or third-party packages blindly
- Automated installs allow malicious updates or replacements
- Attackers compromise the chain: registry, package, or maintainer

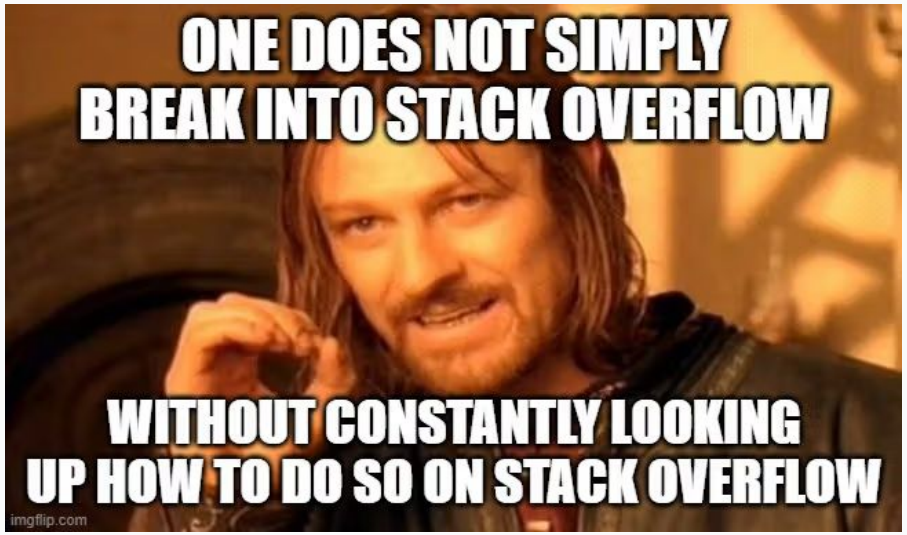
SecAppDev

3. Dependency Chain Abuse - Best practices

- Pin versions and lock dependencies (package-lock.json, requirements.txt)
- Use only vetted and trusted registries
- Sign and verify package integrity (e.g., Sigstore, Cosign)
- Monitor for compromise of upstream dependencies
- Scan and audit dependencies continuously

SecAppDev

2. Inadequate Identity and Access Management - Real World Case: Stack Overflow's Exposed TeamCity Server (2021)



Summary:

- Stack Overflow operated a TeamCity build server that was accessible from the internet.
- The server was misconfigured, allowing unauthorized access without proper authentication.
- Attackers exploited this exposure to gain access to the build environment.

Impact:

- Potential compromise of the CI/CD pipeline.
- Risk of unauthorized code execution and access to sensitive information.
- Necessitated a comprehensive security review and remediation efforts.

Source: https://stackoverflow.blog/2021/01/25/a-deeper-dive-into-our-may-2019-security-incident/

SecAppDev

2. Inadequate Identity and Access Management - Risk Explained

- Weak or missing authentication on CI/CD services
- Over-permissioned users, tokens, service accounts
- Inconsistent IAM across tools: Git, CI, registries, cloud

SecAppDev

2. Inadequate Identity and Access Management - Best practices

- Use centralized IAM and SSO across all pipeline tools
- Enforce least privilege for users and service accounts
- Rotate tokens and credentials regularly
- Audit IAM policies and access logs
- Prefer short-lived, scoped tokens (e.g. OIDC federation)

SecAppDev

1. Insufficient Flow Control Mechanisms
- Real World Case: PHP Git Server Compromise (2021)

Summary:

- In March 2021, attackers pushed two malicious commits to the PHP source code repository, masquerading as legitimate contributors.
- The commits introduced a backdoor that allowed remote code execution when a specific HTTP header was present.
- The malicious code was disguised as a minor typo fix to evade detection. OBJ OBJ OBJ

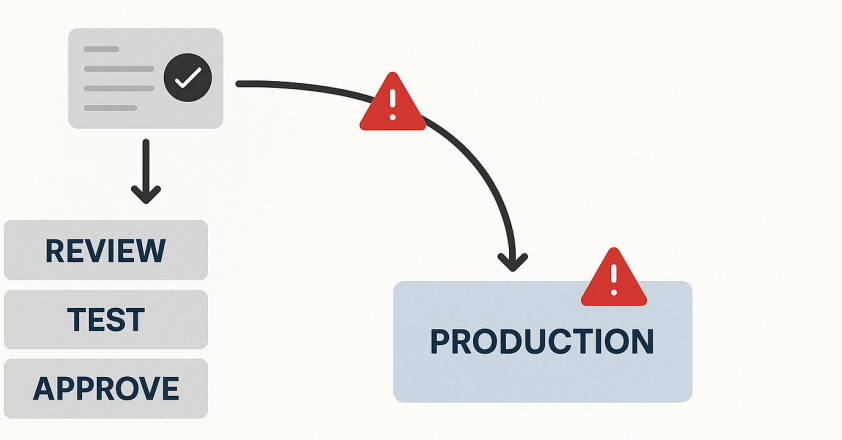
Impact:

- Potential compromise of any server running the tainted PHP version, leading to widespread security risks.
- Erosion of trust in the PHP development process and its infrastructure.

Source: https://news-web.php.net/php.internals/113981

SecAppDev

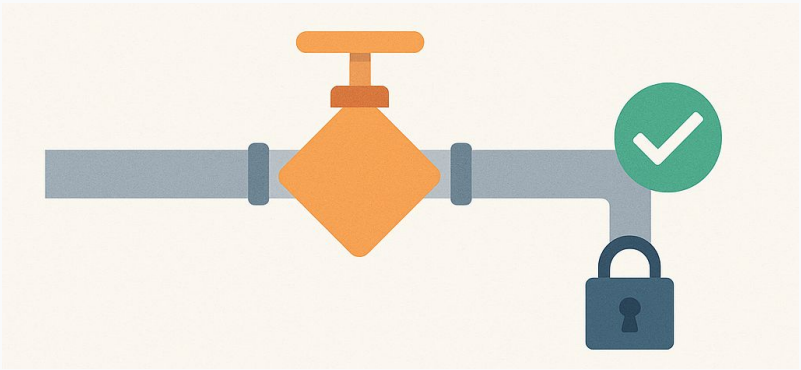
1. Insufficient Flow Control Mechanisms - Risk Explained



- Anyone can trigger builds, merges, or deploys
- Lack of approvals for critical steps (e.g. production deploy)
- No enforcement of peer review, testing, or artifact promotion flows

SecAppDev

1. Insufficient Flow Control Mechanisms - Best practices



- Require approvals before merge/deploy (e.g. code owners, peer review)
- Gate promotions with test + security checks
- Lock deploys behind change management or handoffs
- Use protected branches, environment rules, and manual approvals
- Audit and enforce the pipeline flow via policy-as-code

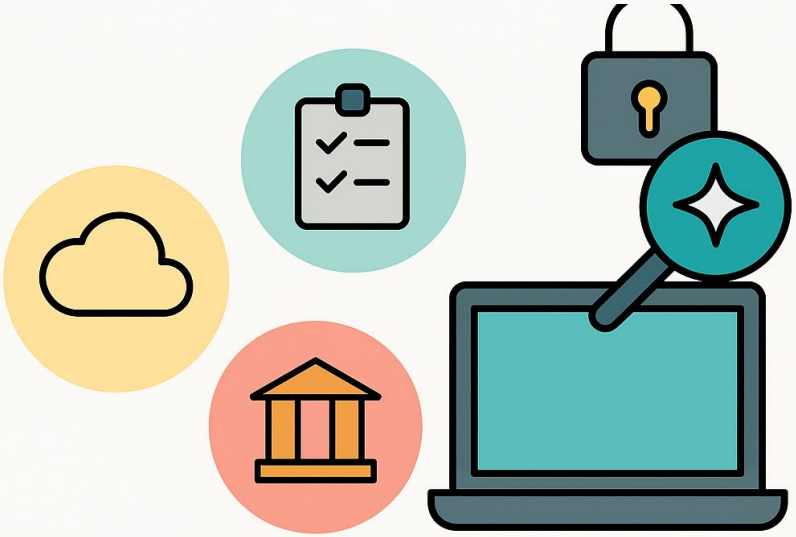
SecAppDev

What's Not in the Top 10 (But Still Matters)

- No mention of SAST, DAST, or container scanning
- Limited focus on toolchain coverage (e.g. IaC, SBOMs)
- Gaps in organizational practices: metrics, SLAs, ownership

SecAppDev

Security Tools to Know



- Wiz
- Aikido
- TruffleHog
- Gitleaks
- Semgrep
- Sigstore / Cosign
- OPA / Conftest
- Dependency Track / OWASP Dependency-Check

SecAppDev

Resources

- OWASP Top 10 CI/CD Security Risks: https://owasp.org/www-project-top-10-ci-cd-security-risks/
- OWASP CI/CD Security Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html

SecAppDev

Thank you

gijs.vanlaer@xfa.tech

SecAppDev