République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Analyse et reproduction contrôlée de la campagne GhostAction

Ce document est réalisé par :

- BAHLOUL Rami

Étudiant en dernière année en cybersécurité, ESTIN, Béjaïa, Algérie

Destiné à :

- Professeur.Benoit Baudry

Table des matières

Liste des tableaux

Introduction générale

Les pipelines d'intégration et de déploiement continus (CI/CD) reposent aujourd'hui fortement sur des workflows déclaratifs " notamment GitHub Actions" pour automatiser la construction, les tests et la distribution du code. Ces workflows, souvent décrits dans des fichiers YAML stockés directement dans les dépôts, disposent d'un accès aux secrets (tokens d'API, clés de publication, identifiants cloud, etc.) qui leur permet d'interagir avec les registres, les services tiers ou l'infrastructure. Cette centralisation des privilèges en fait une cible attractive : un workflow compromis ou mal configuré peut conduire à une fuite massive de secrets et à une compromission étendue.

L'incident documenté par StepSecurity sous le nom de campagne « GhostAction » illustre ce risque : des workflows malveillants ont été utilisés pour exfiltrer des secrets depuis des dépôts et runners GitHub, affectant un grand nombre de projets. Le présent travail vise à étudier ce type d'attaque de manière contrôlée et responsable. L'objectif principal est double : d'une part identifier, via une analyse automatisée et manuelle, des workflows potentiellement malicieux sur des dépôts publics ; d'autre part mettre en place un environnement minimal et sûr permettant de reproduire, avec des secrets factices, les mécanismes d'exfiltration observés afin de valider des heuristiques de détection. Cette démarche combine des techniques d'analyse statique (recherche d'indicateurs dans les fichiers YAML), de prétraitement et d'enrichissement des métadonnées, la génération contrôlée de workflows simulés, ainsi que des expérimentations dynamiques dans un environnement isolé (serveur local de capture).

Chapitre 1

Résumé exécutif

Le projet a permis d'implémenter une chaîne expérimentale complète : collecte de workflows, prétraitement, analyses statiques pour repérer des candidats suspects, génération automatique de 7 workflows simulés contenant des étapes d'exfiltration factices, simulation d'exfiltration vers un serveur local Flask et comparaison automatique des captures avec l'analyse statique. Les 7 workflows simulés ont correctement envoyé leurs secrets factices et ont été identifiés comme *Match* par le script de comparaison : la méthode (dans l'environnement contrôlé) s'est révélée efficace.
Contraintes éthiques : toutes les expérimentations sont faites localement, avec des secrets factices et sans exfiltration vers des serveurs publics.

Chapitre 2

Contexte & objectifs

2.1 Contexte

L'article StepSecurity décrit la campagne GhostAction, expliquant des stratégies d'exfiltration via GitHub Actions. Toutefois, l'article n'expose pas tous les fichiers YAML originaux. Le travail consiste donc à rechercher, identifier et expérimenter des workflows candidats sur des dépôts accessibles.

2.2 objectifs

L'objectif de ce travail de recherche est d'identifier et d'analyser les **workflows malicieux impliqués dans l'attaque GhostAction**, documentée par StepSecurity (GhostAction Campaign). L'article fournit des exemples de workflows exfiltrant des secrets GitHub, mais ne liste pas l'ensemble des workflows affectés. Cette recherche vise donc à récupérer des workflows suspects, à les analyser et à préparer les bases pour une simulation expérimentale sécurisée.

2.3 Environnement de travail

- **OS :** Windows (PowerShell) — utilisateur `PC SOLUTION`.
- **Python :** 3.13.7 (`virtualenv .venv`).
- **Librairies principales :** `PyYAML`, `Flask`, `pandas`, `requests`.
- **Outils additionnels :**
  - `act` — simulation locale des GitHub Actions.
  - Docker Desktop — tentative d'utilisation (limitée par absence de virtualisation sur la machine).

- **Dépôt local :** `ghostaction-research/` (structure détaillée en annexe).
- **Scripts principaux (dossier `tools/`) :**
  - `collect_workflows.py` — collecte via GitHub Search API.
  - `preprocess_workflows.py` — validation et normalisation YAML.
  - `analyze_workflows.py` — détection statique et création de `workflow_suspicious.cs`
  - `generate_simulated_workflows.py` — génération des fichiers `ci_simulated1..7.ym`
  - `analyze_visualize_workflows.py` — visualisations et statistiques.
  - `compare_workflows.py`, `compare_captures_static.py` — comparaison statique vs captures.
- **Simulation :**
  - `simulation/capture-server/server.py` — application Flask qui sauvegarde chaque `POST` dans `captures/`.
- **Fichiers CSV produits :**
  - `workflow_summary.csv` — résumé de la collecte.
  - `workflow_suspicious.csv` — workflows identifiés comme suspects (7 workflows retenus).
  - `analysis/capture_vs_static_match.csv` — comparaison captures vs statique (résultats).

Chapitre 3

Méthodologie

3.1 Collecte des workflows

- Construction de requêtes GitHub Search à partir d'indicateurs de compromission (IoC) et mots-clés suspects : `curl`, `wget`, `${{ secrets.}`.
- Utilisation de l'API `search/code` pour lister les fichiers YAML suspects.
- Récupération des blobs (encodés en base64) puis conversion en texte YAML.
- Sauvegarde des fichiers collectés dans le dossier `collected-workflows/`.
- Enregistrement des métadonnées dans `workflow_summary.csv`, avec les champs principaux :
  - `file` : nom du fichier YAML,
  - `workflow_name`,
  - `triggers`,
  - `jobs`,
  - `secrets`,
  - `num_jobs`, `num_secrets`.

3.2 Prétraitement et validation

- Normalisation des fichiers :
  - nettoyage des encodages,
  - remplacement des tabulations,
  - parsing YAML via `yaml.safe_load_all` (acceptation de plusieurs documents YAML par fichier).

- Gestion des cas particuliers :
  - échec de parsing (tags exotiques, templates GitLab, etc.) → fichier marqué comme `invalide`.
  - les workflows valides sont conservés pour l'analyse.

3.3 Organisation et visualisation des données

- Importation des résultats dans un `DataFrame Pandas` pour faciliter les comparaisons.
- Identification automatique des workflows suspects selon le nombre de jobs, l'utilisation de secrets et la présence de commandes d'exfiltration.
- Export des résultats sous forme de CSV :
  - `workflow_summary.csv` : résumé global,
  - `workflow_suspicious.csv` : workflows retenus comme suspects pour la simulation.

Chapitre 4

Analyse statique — Détection des workflows suspects

L'analyse statique a été effectuée sur l'ensemble des workflows collectés et prétraités afin d'identifier les fichiers potentiellement malveillants. Les critères de suspicion utilisés reposent sur des heuristiques simples mais efficaces :

- Présence de références explicites à des `secrets` dans le YAML (`env`, `${{ secrets.}}`).
- Détection de commandes d'exfiltration possibles dans les étapes `run:` (`curl`, `wget`, `nc`, ou encore `python requests.post`).
- Jobs comprenant des étapes inhabituelles pour un pipeline CI classique (ex. écriture de fichiers contenant des secrets, upload externe).

Les résultats de cette détection ont été exportés dans le fichier `workflow_suspicious.csv`. Sept workflows ont été retenus comme candidats suspects.

Fichier YAML	**Description / Rôle**
`.github_gitflow_linters.yml`	Linters
`.github_workflows_nightly_build.yml`	Nightly Build
`.github_workflows_publish.yml`	Publish
`.github_workflows_release_nightly.yml`	Langflow Nightly Bui
`main.yml`	Daily Checks for appc
`out_checkstyle_checkstyle_.github_workflows_diff_report.yml`	DiffReport
`out_sass_sass_.github_workflows_ci.yml`	CI

TABLE 4.1 – Workflows identifiés comme suspects après analyse statique.

Chapitre 5

Génération des workflows simulés

Pour minimiser les risques tout en permettant une reproduction contrôlée des scénarios d'exfiltration, chaque workflow identifié comme suspect a été transformé en un workflow simulé. Ces workflows simulés ont été générés automatiquement et sont stockés dans le répertoire `.github/workflows/` sous les noms `ci_simulated1.yml` à `ci_simulated7.yml`.

5.1 Principes appliqués

— **Conservation de la structure** : la structure logique (nombre de jobs, noms proches des jobs originaux) est conservée pour garder la similarité avec les workflows réels sans exécuter leurs actions potentiellement dangereuses.

— **Remplacement des secrets** : toute référence aux secrets réels a été remplacée par une variable factice `secrets.TEST_SECRET` ou par une variable d'environnement `TEST_SECRET` fournie lors de la simulation.

— **Exfiltration simulée** : chaque workflow inclut une étape finale contrôlée qui envoie le secret factice vers l'endpoint local `/collect` du serveur Flask de capture (simulation/capture-server). L'envoi se fait en priorité vers `http://host.docker.internal:5000/collect` puis en fallback vers `http://127.0.0.1:5000/collect`.

— **Commentaires d'origine** : chaque fichier généré contient un en-tête commenté indiquant le fichier source d'origine et le nom du workflow détecté, pour permettre le traçage et la revue manuelle.

— **Sécurité** : neutre vis-à-vis des données réelles — aucun secret réel n'a été utilisé ou exposé.

5.2 Emplacement et noms des fichiers

Les fichiers générés se trouvent dans :

```
.github/workflows/
   ci_simulated1_linters.yml
   ci_simulated2_nightly-build.yml
   ci_simulated3_publish.yml
   ci_simulated4_langflow-nightly-build.yml
   ci_simulated5_daily-checks-for-appcodeecommerce121.yml
   ci_simulated6_diffreport.yml
   ci_simulated7_ci.yml
```

5.3 Script d'automatisation

Le script utilisé (`tools/generate_simulated_workflows.py`) lit le fichier `workflow_suspicious.csv` et génère automatiquement les YAML simulés. Chaque YAML contient :

— un header commenté indiquant la source,

— des jobs proches de l'original,

— une étape finale qui poste le secret factice vers `/collect`.

5.4 Secrets factices utilisés

Pour faciliter le suivi des captures, un secret factice distinct a été utilisé par workflow. Exemples :

```
dummy_secret_value_123
dummy_LIN_123
dummy_NB_123
dummy_PUB_123
dummy_LF_123
dummy_DC_123
dummy_DR_123
```

5.5 Remarques

— L'approche permet de reproduire un scénario d'exfiltration de secrets sans toucher à des données sensibles réelles.

Chapitre 6

Simulation & capture (serveur local)

6.1 Serveur de capture

- **Fichier :** `simulation/capture-server/server.py` (Flask).
- **Fonction :** reçoit des requêtes HTTP `POST` sur l'endpoint `/collect` et sauvegarde chaque payload sous forme JSON (métadonnées, en-têtes, corps, timestamp) dans le répertoire `simulation/capture-server/captures/`.
- **Objectif :** fournir un endpoint local contrôlé qui permet de vérifier, en environnement isolé, si un workflow (ou sa simulation) exfiltre des secrets vers un serveur externe.
- **Format de sortie :** chaque capture est un fichier JSON contenant au minimum les champs `received_at`, `remote_addr`, `headers`, `body` et `json` (si le corps est du JSON).

Exemple de capture (extrait)


```
{
  "received_at": "2025-09-13T16:03:12.847178Z",
  "remote_addr": "127.0.0.1",
  "headers": {
    "User-Agent": "...",
    "Content-Type": "application/json",
    ...
  },
  "body": "{ \"secret\": \"dummy_secret_value_123\", \"workflow\": \"ci_simulate
  "json": {
```

```
    "secret": "dummy_secret_value_123",
    "workflow": "ci_simulated7_ci"
  }
}
```

6.2 Exécution des simulations

— **Méthodes testées**

1. `act` + **Docker** : objectif initial exécuter les workflows simulés localement dans des conteneurs proches des runners GitHub Actions.
2. **Exécution manuelle** : scripts PowerShell / `curl` simulant uniquement les étapes d'exfiltration présentes dans les workflows générés. Cette méthode n'exécute pas l'intégralité des jobs (build/-tests), mais reproduit fidèlement la logique d'envoi des secrets factices vers `/collect`.

— **Résultat** : exécution des 7 simulations (un envoi par workflow) aboutissant à la réception et la sauvegarde de 7 captures JSON dans `simulation/capture-server/`. Les payloads contiennent les secrets factices et un identifiant de workflow, permettant d'associer chaque capture au fichier simulé correspondant.

Chapitre 7

Comparaison captures ↔ analyse statique - résultats

7.1 Méthodologie de comparaison

La dernière étape du pipeline consiste à confronter les résultats issus de l'**analyse statique** (identification de workflows suspects) avec ceux de la **simulation dynamique** (captures réellement exfiltrées par le serveur local).

- **Script utilisé :** `tools/compare_captures_static.py`.
- **Principe de fonctionnement :**
  1. Charger le fichier `workflow_suspicious.csv` (résultats de l'analyse statique).
  2. Parcourir tous les artefacts de capture générés dans `simulation/capture-server/cap`
  3. Pour chaque capture, lire le champ `workflow` (nom du workflow qui a envoyé un secret factice).
  4. Normaliser le nom du workflow (minuscules, suppression des extensions et chemins) afin d'éviter les faux négatifs dus à des différences de format.
  5. Vérifier si le workflow est présent dans la liste des workflows suspects.
  6. Si oui ⇒ **Match**, sinon ⇒ **NoMatch**.
- **Sortie produite :** un fichier `analysis/capture_vs_static_match.csv` listant, pour chaque capture :
  - le nom du fichier suspect initial,
  - le workflow simulé exécuté,
  - le secret capturé (factice),

— le résultat (`Match` ou `NoMatch`),
— des détails éventuels (ex. longueur du secret).

7.2 Résultats obtenus

Tous les 7 workflows simulés ont produit une exfiltration correcte vers le serveur Flask. Le fichier `capture_vs_static_match.csv` indique donc 7 lignes marquées `Match`.

Workflow statique	**Secret capturé**	**Résultat**
Linters	`dummy_LIN_123`	Match
Nightly Build	`dummy_NB_123`	Match
Publish	`dummy_PUB_123`	Match
Langflow Nightly Build	`dummy_LF_123`	Match
Daily Checks (appcodeEcommerce121)	`dummy_DC_123`	Match
DiffReport	`dummy_DR_123`	Match
CI	`dummy_secret_value_123`	Match

TABLE 7.1 – Résumé des correspondances statique ↔ simulation.

7.3 Interprétation

- L'analyse statique, basée sur des heuristiques simples (détection de secrets, présence de commandes suspectes), a correctement identifié les 7 workflows problématiques.
- La simulation contrôlée confirme que chacun de ces workflows, une fois exécuté, exfiltre effectivement un secret (factice) vers l'extérieur.
- L'absence de `NoMatch` montre que la chaîne complète (collecte → prétraitement → détection statique → génération de workflows simulés → capture → comparaison) est **cohérente et opérationnelle**.
- Cela valide le choix méthodologique : utiliser des secrets factices et un serveur de capture local permet de démontrer l'efficacité de l'attaque GhostAction dans un environnement sécurisé et reproductible.

Chapitre 8

Problèmes rencontrés et solutions

8.1 Problèmes techniques et solutions

1. **Execution Policy PowerShell**
   Problème : l'exécution de `Activate.ps1` était bloquée.
   Solution : lancer PowerShell en mode administrateur ou ajuster la policy avec :

   ```
   Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
   ```

2. **Parsing YAML variés**
   Problème : certains fichiers collectés (ex. GitLab CI, cloud-init, templates) comportaient des tags ou anchors non compatibles.
   Solution : ignorer ou consigner les fichiers non-parsables et continuer l'analyse sans bloquer la collecte.
3. **Comparaison automatique (NoMatch)**
   Problème : différence de normalisation des noms (workflow vs. fichier YAML).
   Solution : appliquer une normalisation (minuscules, suppression des espaces/caractères spéciaux) ou réaliser un mapping manuel via le CSV. Après correction, la comparaison renvoie *Match*.
4. **Pare-feu Windows**
   Problème : Windows Defender bloquait certaines connexions Python locales.
   Solution : autoriser la communication locale (`127.0.0.1` uniquement) pour les tests.

8.2 Limites rencontrées

1. **Informations manquantes** : Certains fichiers YAML n'indiquaient pas de manière exhaustive les secrets ou triggers.
2. **Accès externe impossible** : Les appels à certaines API pour enrichir la collecte ont échoué (limitations réseau).
3. **Sécurité des secrets** : Obligation d'utiliser uniquement des secrets factices afin d'éviter toute fuite accidentelle.
4. **Première expérience de recherche** : Comme il s'agit de mon premier projet de recherche, la phase initiale a été difficile : je ne comprenais pas clairement comment commencer et structurer le travail. L'avancement progressif, ainsi que l'analyse d'exemples existants, m'ont aidé à surmonter cette difficulté.

Chapitre 9

Conclusions & enseignements

La méthode suivie collecte → analyse statique → simulation contrôlée → capture → comparaison s'est révélée opérationnelle. Elle permet de reproduire, dans un environnement sûr et contrôlé, le type d'exfiltration décrit dans la campagne GhostAction.

9.1 Efficacité de la simulation

Tous les secrets factices injectés lors des simulations ont été correctement capturés par le serveur local de collecte. Les correspondances entre l'analyse statique (workflows identifiés comme suspects) et les captures dynamiques sont exactes pour les sept workflows retenus, ce qui prouve la cohérence du pipeline d'analyse et de test mis en place.

9.2 Valeur pédagogique et utilité

Cette reproduction contrôlée offre plusieurs apports :

— visualiser le flux d'un secret depuis un workflow CI jusqu'à sa potentielle exfiltration ;

— fournir un banc d'essai pour évaluer des outils de détection (analyse statique et dynamique) ;

— sensibiliser au danger d'exposer des secrets dans des workflows et à l'importance de bonnes pratiques (rotation, scope minimal, revue des modifications dans `.github/workflows/`).

9.3 Limites

— Les simulations utilisent des **secrets factices** : le comportement réel en production peut différer (obfuscation des commandes, étapes conditionnelles, runners auto-hébergés avec accès réseau différent, etc.).

— Les heuristiques d'analyse statique peuvent générer des *faux positifs* et *faux négatifs* — elles doivent être affinées et complétées par des analyses plus profondes.

— L'environnement de test local simplifie certains aspects réseau (par ex. résolution `host.docker.internal` vs. runner distant) — pour des tests proches de la réalité, l'emploi de conteneurs Docker/VMs et d'un réseau isolé est recommandé.

9.4 Bilan

Le pipeline développé constitue une base solide pour poursuivre la recherche sur GhostAction : il permet d'itérer rapidement (nouveaux IoC, nouvelles règles), d'ajouter des jeux de données réels contrôlés et d'automatiser des scénarios d'attaque/détection. Pour aller plus loin, il faudra élargir les jeux de tests, intégrer des scanners CI/CD commerciaux ou open-source et exécuter des simulations dans des environnements plus proches de la production.

Chapitre 10

Annexes

10.1 Structure du dépôt (extrait)

```
ghostaction-research/
 .venv/
 .github/workflows/ (ci_simulated*.yml)
 collected-workflows/
 tools/
   collect_workflows.py
   preprocess_workflows.py
   analyze_workflows.py
   generate_simulated_workflows.py
   analyze_visualize_workflows.py
   compare_captures_static.py
 simulation/
   capture-server/
     server.py
     captures/
 analysis/
   capture_vs_static_match.csv
 workflow_summary.csv
 workflow_suspicious.csv
 reports/
    Ghostaction_Rapport.pdf
    GhostAction_préliminaire.pdf
```

10.2 Commandes utiles

```
# Activer venv (PowerShell)
.venv\Scripts\Activate.ps1

# Lancer le serveur de capture (dans simulation/capture-server/)
python server.py

# Générer workflows simulés
python tools/generate_simulated_workflows.py

# Comparer captures vs statique
python tools/compare_captures_static.py

# Analyser / visualiser
python tools/analyze_visualize_workflows.py
```

10.3 Extrait de capture (exemple)


```
{
  "received_at": "2025-09-13T16:03:12.847178Z",
  "remote_addr": "127.0.0.1",
  "json": {
    "secret": "dummy_secret_value_123",
    "workflow": "ci_simulated7_ci"
  }
}
```

Conclusion générale

Ce projet a constitué ma première véritable expérience de recherche appliquée dans le domaine de la cybersécurité. Dès le début, j'ai été confronté à des difficultés méthodologiques et techniques, notamment pour comprendre comment structurer l'approche et poser les premières étapes d'analyse. Progressivement, en avançant par itérations, j'ai pu construire un pipeline cohérent allant de la collecte de workflows GitHub jusqu'à la simulation et la comparaison des résultats.

Sur le plan pratique, ce travail m'a permis de manipuler des outils variés (API GitHub, Python, Flask, analyse statique de YAML, génération de workflows simulés) et de mettre en place un environnement de test sécurisé. J'ai observé que la simulation contrôlée, même avec des secrets factices, permet de démontrer de manière concrète les risques liés à l'exfiltration dans les workflows CI/CD. Cette visualisation pratique m'a donné une meilleure compréhension du fonctionnement interne des pipelines CI et des vulnérabilités possibles.

Sur le plan réflexif, j'ai pris conscience de l'importance d'adopter une démarche scientifique rigoureuse : définir un protocole clair, documenter les étapes, gérer les erreurs et ajuster la méthodologie au fur et à mesure. Ce projet m'a également sensibilisé à la nécessité de trouver un équilibre entre analyse statique (rapide mais approximative) et analyse dynamique (plus fidèle mais plus lourde à exécuter).

En définitive, cette expérience m'a apporté non seulement des compétences techniques mais aussi une meilleure méthodologie de travail et de recherche. Elle a posé les bases pour mes futurs projets : approfondir l'étude des attaques contre les chaînes CI/CD, explorer des environnements plus réalistes et contribuer à des solutions de détection et de prévention.

Remerciements

Merci à Prof. Baudry pour la proposition du sujet et l'encadrement. Ce travail est une reproduction contrôlée et éducative destinée à comprendre les vecteurs d'exfiltration par workflows CI/CD.

Bibliographie

[1] StepSecurity. *GhostAction Campaign : Over 3,000 Secrets Stolen Through Malicious GitHub Workflows*. Disponible en ligne : https://www.stepsecurity.io/blog/ghostaction-campaign-over-3-000-secrets-stolen-through-malicious-github-worki consulté en septembre 2025.

[2] GitGuardian. *GhostAction Campaign — 3,325 Secrets Stolen*. Disponible en ligne : https://blog.gitguardian.com/ghostaction-campaign-3-325-secrets-stolen/, consulté en septembre 2025.

[3] GitHub Documentation. *Understanding GitHub Actions Workflows*. Disponible en ligne : https://docs.github.com/en/actions/using-workflows, consulté en 2025.

[4] Wes McKinney. *Data Analysis with Python and Pandas*. O'Reilly Media, 2017.

[5] Armin Ronacher. *Flask Documentation*. Disponible en ligne : https://flask.palletsprojects.com, consulté en 2025.

[6] nektos/act. *Run your GitHub Actions locally*. Disponible en ligne : https://github.com/nektos/act, consulté en 2025.

Date : 14 septembre 2025