

Rapport de test d'intrusion — OWASP Top 10 2025

A09:2025 — Security Logging & Alerting Failures

Cible	VulnLog (application web de lab, <code>http://127.0.0.1:5009</code>)
Environnement	Pile Docker isolée : app + Promtail + Loki + Grafana + fail2ban (bind <code>127.0.0.1</code> uniquement)
Type d'évaluation	Boîte grise — lab pédagogique, autorisé
Catégorie OWASP	A09:2025 — Security Logging & Alerting Failures
CWE couverts	CWE-778, CWE-223, CWE-221, CWE-117, CWE-532
Groupe	#9 (numéro de vulnérabilité = numéro de groupe)
Date	2026-07-04
Statut global	Élevé — amplificateur de risque : compromission non détectée , aucune réponse à incident possible

1. Résumé exécutif

A09 n'est **pas** une faille d'injection : c'est l'**incapacité à détecter une intrusion et à y répondre**, faute de journalisation et d'alerte suffisantes. OWASP le résume ainsi : « sans journalisation ni supervision, les attaques et les compromissions ne peuvent pas être détectées ». C'est une faille **transverse** qui amplifie toutes les autres : une attaque réussie (brute-force, A01, injection...) reste **invisible**, ce qui allonge le temps de présence de l'attaquant (dwell time) et rend l'investigation impossible.

En 2025, la catégorie a été **renommée** (« Monitoring » → « **Alerting** ») pour insister sur le fait que journaliser ne suffit pas : encore faut-il **alerter** et réagir. Elle est votée par la communauté pour la 3^e fois consécutive (incidence moyenne **3,91 %**, max **11,33 %**, **723 CVE** associées).

Sur l'application **VulnLog**, une attaque réaliste — **30 tentatives de brute-force** sur `admin`, **énumération de comptes, accès non autorisé** à une fonction sensible, puis **compromission du compte admin** — a été menée dans deux configurations :

	Mode vulnérable (A09 présent)	Mode corrigé (outil déployé)
Journal de sécurité	vide	

	Mode vulnérable (A09 présent)	Mode corrigé (outil déployé)
		37 évènements structurés (IP, user, résultat)
Centralisation (Loki)	0 évènement	37 évènements , dont 35 LOGIN_FAILURE
Détection (fail2ban)	rien	30 échecs détectés , IP 10.66.66.66 bannie
Alerte	aucune	BRUTE_FORCE_LOGIN déclenchée

Verdict : en mode vulnérable, l'attaquant compromet le compte admin **sans laisser la moindre trace exploitable**. Après déploiement de l'outil de supervision, la **même attaque** est **journalisée, centralisée, détectée et alertée** — A09 est corrigée et la correction est **vérifiée**.

2. La vulnérabilité A09 en détail

2.1 Description

L'application traite les évènements sensibles (authentification, contrôle d'accès) mais **ne les journalise pas de façon exploitable** :

- les **échecs d'authentification** ne sont pas tracés → un brute-force ne déclenche rien ;
- aucun **contexte** de sécurité n'est enregistré (IP source, utilisateur visé, résultat) ;
- il n'existe **aucun mécanisme d'alerte** : même un pic d'échecs passe inaperçu ;
- les seuls logs présents (accès HTTP génériques) sont **inutilisables** pour la sécurité.

Piège classique : « on a des logs » ≠ « on a des logs de sécurité exploitables et surveillés ».

2.2 Mapping CWE

CWE	Intitulé	Présence dans VulnLog (mode vulnérable)
CWE-778	Insufficient Logging	Aucun log des échecs d'authentification ni des accès refusés
CWE-223	Omission of Security-relevant Information	Ni IP source, ni utilisateur, ni résultat journalisés
CWE-221	Information Loss / Omission	Perte de l'information nécessaire à la détection
CWE-117	Improper Output Neutralization for Logs	Champs contrôlés par l'attaquant non neutralisés (risque d'injection de logs)

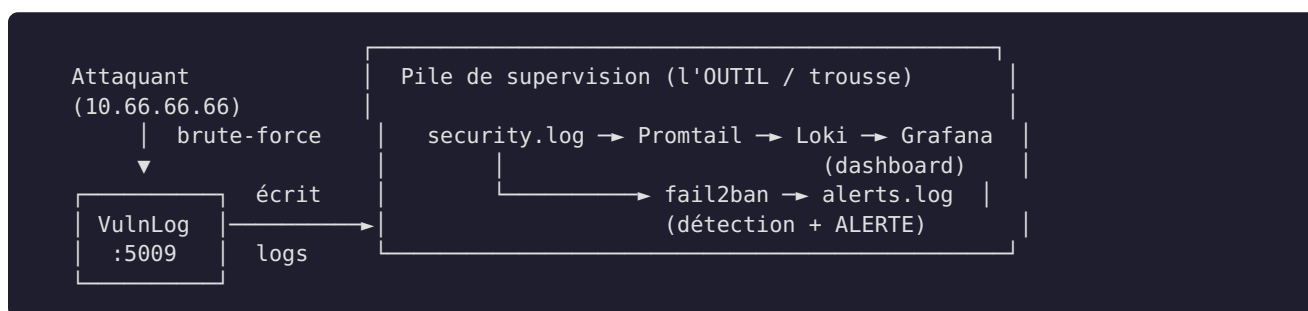
CWE	Intitulé	Présence dans VulnLog (mode vulnérable)
CWE-532	Insertion of Sensitive Information into Log File	Risque de journaliser des secrets (mots de passe) — à éviter

2.3 Impact

- **Détection nulle** : compromission invisible, dwell time qui explose (moyenne secteur > 200 jours).
- **Réponse à incident impossible** : sans traces, pas de périmètre, pas de forensic, pas de preuve.
- **Conformité** : violation des exigences de journalisation (ISO 27001 A.8.15/A.8.16, PCI-DSS 10, RGPD art. 33 notification sous 72 h).

3. Périmètre & environnement

3.1 Architecture du lab



3.2 Services (tout bindé sur 127.0.0.1)

Service	Rôle	URL
VulnLog (Flask)	Application cible	<code>http://127.0.0.1:5009</code>
Promtail	Collecte des journaux	—
Loki	Centralisation / requêtage (LogQL)	<code>http://127.0.0.1:3109</code>
Grafana	Visualisation (dashboard fourni)	<code>http://127.0.0.1:3009</code>
fail2ban	Détection brute-force + alerte	—

3.3 Mécanisme de démonstration

La même application tourne dans deux modes, pilotés par la variable d'environnement `SECURE_LOGGING` — ce qui permet de comparer **avant/après** avec exactement la même attaque :

<code>SECURE_LOGGING</code>	Comportement
<code>0</code> (défaut)	Vulnérable — aucun évènement de sécurité émis (défaut A09)

SECURE LOGGING	Comportement
1	Corrigé — évènements de sécurité structurés + centralisés + surveillés

4. Détection de la vulnérabilité — avec quel outil ?

4.1 Outils utilisés

- **Générateur d'attaque** : script `run_attacks.sh` (curl) simulant un outil de brute-force type **Hydra** (User-Agent: hydra/9.5 , IP attaquant via X-Forwarded-For).
- **Instrument d'audit** : la pile **Promtail + Loki + Grafana** (interrogée en **LogQL**) et **fail2ban** (`fail2ban-client status`).

4.2 Procédure

```
cd a09-lab
bash run_demo.sh vuln      # démarre en SECURE_LOGGING=0 + lance l'attaque + audite
```

Le principe de détection de A09 : **on lance une attaque connue et on audite la chaîne de supervision**. Si l'attaque ne produit **ni évènement de sécurité ni alerte**, la vulnérabilité A09 est **confirmée**.

4.3 Preuve — l'attaque est totalement invisible (mode vulnérable)

Attaque exécutée (extrait) :

```
>> 1. Brute-force d'authentification – 30 tentatives sur 'admin'
tentative 1 -> HTTP 401
...
tentative 30 -> HTTP 401
>> 3. Accès non autorisé à une fonction sensible
GET /api/admin -> HTTP 403
>> 4. (bonus) login enfin réussi – compromission du compte admin
{"ok":true,"user":"admin"}
```

Audit de la supervision **immédiatement après** l'attaque :

```
### security.log (mode vulnérable) – 0 ligne(s)

### Loki – évènements de sécurité LOGIN_FAILURE :
==> 0 évènement(s) de sécurité dans Loki

### fail2ban – statut du jail :
|- Filter
|  |- Currently failed: 0
|  |- Total failed:    0
`- Actions
   |- Currently banned: 0
   `- Banned IP list:

### alerts.log – vide
```

A09 confirmée : 30 échecs d'authentification + une compromission réussie, et **aucune trace, aucune détection, aucune alerte**. L'attaquant est indétectable.

5. Correction de la vulnérabilité — avec quel outil ?

La correction est en **deux volets** complémentaires : (a) l'application doit **produire** l'évènement de sécurité, (b) l'outil de supervision doit le **centraliser, détecter et alerter**.

5.1 Volet applicatif — journalisation structurée

Code fautif (avant) — l'échec d'authentification ne journalise rien :

```
@app.route("/api/login", methods=["POST"])
def login():
    ...
    if USERS.get(user) == pw:
        return jsonify({"ok": True, "user": user})
    return jsonify({"ok": False, "error": "invalid credentials"}), 401
    # x échec non journalisé : ni IP, ni user, ni résultat -> rien à alerter
```

Correction — chaque évènement de sécurité est émis en **logfmt structuré** (horodatage UTC, `event`, `src_ip`, `user`, `outcome`, `path`, `ua`), vers un flux dédié `security.log`. Le **mot de passe n'est jamais journalisé** (CWE-532), les champs contrôlés par le client sont neutralisés (CWE-117) :

```
def security_event(event, outcome, user="-"):
    line = (f"{now()} level=SECURITY event={event} "
           f"src_ip={client_ip()} user={_q(user)} outcome={outcome} "
           f"path={request.path} method={request.method} ua={_q(ua)}")
    security_log.info(line) # -> security.log -> Promtail -> Loki ; surveillé par fail2ban

# dans login():
    security_event("LOGIN_FAILURE", "failure", user) # ✓ échec tracé ET alertable
```

5.2 Volet outillage — centralisation + détection + alerte

- **Promtail** collecte `security.log` et le pousse vers **Loki** (centralisation, requêtage LogQL, rétention).
- **Grafana** visualise les évènements (dashboard `A09 – Logging & Alerting`).
- **fail2ban** surveille `security.log` : au-delà de **5 échecs en 60 s** depuis une même IP, il **détecte** le brute-force et **émet une alerte**.

Règle de détection fail2ban (extrait) :

```
# filter.d/vulnlog.conf
failregex = level=SECURITY event=LOGIN_FAILURE src_ip=<HOST>\b

# jail.d/vulnlog.conf
[vulnlog-auth]
logpath = /var/log/vulnlog/security.log
```

```
maxretry = 5
findtime = 60
action   = alert-log[name=vulnlog-auth] # écrit l'alerte dans alerts.log
```

5.3 Procédure

```
bash run_demo.sh fix # recompile l'app en SECURE_LOGGING=1 + relance la MÊME attaque +
audite
```

5.4 Preuve — la même attaque est journalisée, détectée et alertée (mode corrigé)

a) Évènements de sécurité structurés dans `security.log` / centralisés dans Loki (extrait) :

```
2026-07-04T11:21:09Z level=SECURITY event=LOGIN_FAILURE src_ip=10.66.66.66 user=admin
outcome=failure path=/api/login method=POST ua="hydra/9.5 (brute-force)"
2026-07-04T11:21:10Z level=SECURITY event=LOGIN_FAILURE src_ip=10.66.66.66 user=jenkins
outcome=failure path=/api/login method=POST ua="hydra/9.5 (brute-force)"
2026-07-04T11:21:10Z level=SECURITY event=ACCESS_DENIED src_ip=10.66.66.66 user=attacker
outcome=denied path=/api/admin method=GET ua="hydra/9.5 (brute-force)"
2026-07-04T11:21:10Z level=SECURITY event=LOGIN_SUCCESS src_ip=10.66.66.66 user=admin
outcome=success path=/api/login method=POST ua="hydra/9.5 (brute-force)"
...
==> 37 évènement(s) de sécurité dans Loki | LOGIN_FAILURE comptés = 35
```

b) Détection + bannissement par fail2ban :

```
### fail2ban - statut du jail (détection) :
|- Filter
| |- Currently failed: 0
| |- Total failed: 30
`- Actions
  |- Currently banned: 1
  |- Total banned: 1
  `-- Banned IP list: 10.66.66.66
```

c) Alerte déclenchée (`alerts.log`) :

```
2026-07-04T11:21:10Z level=ALERT type=BRUTE_FORCE_LOGIN jail=vulnlog-auth src_ip=10.66.66.66
failures=5 action=DETECTED
```

A09 corrigée et vérifiée : la même attaque produit désormais **37 évènements structurés centralisés**, une **détection** (IP bannie) et une **alerte** brute-force.

6. Procédure de mise en œuvre de l'outil (pour la trousse à outils)

Objectif : permettre à **tout groupe** d'ajouter cette capacité logging + détection + alerte à son projet en < 5 minutes.

1. **Récupérer le lab** puis démarrer la pile : `bash cd a09-lab docker compose up -d --build # app + Promtail + Loki + Grafana + fail2ban`
2. **Brancher l'application** : écrire les événements de sécurité (auth, accès refusé, action sensible) en **logfmt** dans un fichier dédié (`security.log`) — un événement par échec/succès, **avec IP source, utilisateur et résultat, sans secret**.
3. **Collecter** : Promtail `tail security.log` → Loki (`monitoring/promtail-config.yml`).
4. **Visualiser** : Grafana, datasource Loki provisionnée, dashboard `A09 – Logging & Alerting` (`http://127.0.0.1:3009`).
5. **Détecter + alerter** : fail2ban surveille `security.log` (`maxretry=5` , `findtime=60`) et écrit une alerte dans `alerts.log` . En production, remplacer/compléter l'action par un **bannissement iptables**, un **e-mail** ou un **webhook** (Slack/Teams/Alertmanager).
6. **Vérifier** : `bash bash run_demo.sh fix docker exec a09-fail2ban fail2ban-client status vulnlog-auth # -> IP bannie cat logs/alerts.log # -> alerte`

Montée en gamme (production) : la même logique se transpose vers un **SIEM** — Wazuh (règles + tableau de bord + alerting intégrés), la stack **ELK/Elastic Security**, ou **Grafana Loki + Alertmanager**. Le principe A09 reste identique : l'application émet des événements de sécurité exploitables, la plateforme les centralise, détecte et alerte, un playbook de réponse est défini.

7. Vérification — synthèse avant / après

Contrôle	Mode vulnérable	Mode corrigé	Preuve
Journalisation des échecs d'auth	✗ 0	✓ 35 <code>LOGIN_FAILURE</code>	<code>security.log</code> , Loki
Contexte (IP, user, résultat)	✗ absent	✓ complet	logfmt structuré
Centralisation	✗ 0	✓ 37 événements	Loki (LogQL)
Détection brute-force	✗ non	✓ 30 échecs, IP bannie	<code>fail2ban-client status</code>
Alerte	✗ aucune	✓ <code>BRUTE_FORCE_LOGIN</code>	<code>alerts.log</code>

8. Recommandations (alignées OWASP A09:2025)

Priorité	Recommandation
P0	Journaliser toutes les tentatives de connexion, échecs de contrôle d'accès et erreurs de validation, avec contexte (horodatage, IP, utilisateur, résultat).
P0	Les contrôles de sécurité génèrent un log quel que soit le résultat (succès et échec).
P0	Mettre en place alerte + playbook de réponse (pas seulement du log) — c'est le sens du renommage 2025.
P1	Ne jamais journaliser de secret (mot de passe, jeton) — CWE-532.
P1	Neutraliser/encoder les données journalisées issues du client (anti log injection) — CWE-117.
P1	Centraliser les journaux (Loki/ELK/SIEM) hors de la machine attaquable, avec rétection et intégrité (append-only).
P2	Déployer des honeytokens pour piéger l'attaquant ; adopter un cadre de réponse (NIST 800-61r2).

9. Captures d'écran à joindre (exigence du sujet)

1. **Vulnérabilité détectée** (mode `vuln`) : le terminal montrant l'attaque (30×401 + compromission) **et** l'audit renvoyant `security.log` vide, Loki 0 événement, `fail2ban ... 0 banned` → l'attaque est invisible.
2. **Correction avec l'outil** (mode `fix`) : le **dashboard Grafana** A09 – Logging & Alerting (`http://127.0.0.1:3009/d/a09-vulnlog`) montrant les `LOGIN_FAILURE` + la sortie `fail2ban-client status vulnlog-auth` (IP `10.66.66.66` bannie) + la ligne d'alerte `alerts.log`.

Astuce : relancer `bash run_demo.sh fix` juste avant la capture Grafana, puis régler la fenêtre temporelle sur Last 15 minutes et l'auto-refresh sur 5 s.

10. Conclusion

La démonstration illustre le cœur de **A09:2025** : ce n'est pas une faille exploitable en soi, mais un **défaut de visibilité** qui rend toutes les autres attaques **indétectables**. En mode vulnérable, une compromission complète (brute-force → accès admin) ne laisse **aucune trace** ; le simple fait d'**émettre des événements de sécurité structurés** et de les **centraliser + surveiller + alerter** transforme la même attaque en un incident **détecté, tracé et alerté**.

Principe directeur : **journaliser les bons événements, avec le bon contexte, centralisés — et surtout alerter et savoir réagir.**

Lab isolé — bind `127.0.0.1` uniquement. Détruire après usage : `docker compose down -v --rm local`.