ChatGPT

Step 5 – Integrazione della gestione della sicurezza (Secure SDLC)

Analisi dello stato attuale

Il repository Evo-Tactics non contiene documentazione esplicita di threat modeling o risk assessment, né script dedicati alla scansione di sicurezza. Le pipeline CI/CD, pur solide, si concentrano su build, test e validazioni funzionali; non sono presenti controlli per vulnerabilità di dipendenze, analisi statica di sicurezza, rilevamento di segreti o procedure di risposta agli incidenti. Non vi è traccia di misure per la protezione dei dati dei giocatori o per l'adempimento del GDPR.

Le best practice per lo sviluppo sicuro del software suggeriscono di integrare la sicurezza in tutte le fasi del ciclo di vita attraverso analisi dei rischi, threat modeling, controlli di sicurezza automatizzati e pianificazione della risposta agli incidenti [1]. È quindi necessario definire un Secure Software Development Lifecycle (Secure SDLC) adattato al progetto e al team.

Proposte di integrazione della sicurezza

1. **Definizione di una politica di sicurezza e privacy**
2. **Politica di sicurezza interna**: redigere un documento che definisca gli obiettivi di sicurezza (riservatezza, integrità, disponibilità), le responsabilità del team e le normative applicabili (es. GDPR per la gestione dei dati dei giocatori).
3. **Policy di gestione dei segreti**: stabilire linee guida su come gestire API key, token e credenziali, prevedendo l'uso di GitHub Secrets e `.env` non versionati.

4. **Modello di minacce**: creare un threat model che identifichi superfici di attacco (servizi web, database di telemetria, file di configurazione) e mitigazioni. Può essere un diagramma con attori, risorse e flussi di dati.

5. **Analisi dei rischi e threat modeling** [1]

6. **Risk assessment periodico**: definire un processo per valutare periodicamente i rischi e classificare le vulnerabilità in base a impatto e probabilità. Utilizzare checklist o strumenti come OWASP SAMM per strutturare l'analisi.
7. **Threat modeling all'inizio di ogni feature**: prima dello sviluppo di nuove funzionalità (es. telemetria, overlay HUD) redigere uno scenario di minacce e assicurarsi che le contromisure siano implementate (es. rate-limiting, autenticazione).

8. **Tracciabilità**: documentare i rischi identificati e lo status delle mitigazioni in un registro (es. file `docs/security/threat_register.md`).

9. **Automazione dei controlli di sicurezza nella CI/CD**

10. **SAST (Static Application Security Testing)**: integrare strumenti come `bandit` per Python e `eslint --plugin security` o `npm audit` per il codice TypeScript. Questi strumenti

analizzano il codice sorgente e segnalano pattern vulnerabili (injection, uso improprio di librerie, etc.).
11. **Dependency scanning**: aggiungere step a GitHub Actions che eseguano `pip-audit` per le dipendenze Python e `npm audit` per pacchetti Node. È utile anche configurare GitHub Dependabot per aggiornare automaticamente le dipendenze vulnerabili.
12. **Secret scanning**: configurare `truffleHog` o `gitleaks` per scansionare il repository alla ricerca di segreti accidentalmente committati. GitHub offre una funzionalità di secret scanning integrata che può essere abilitata.
13. **Analisi Code QL**: attivare l'azione GitHub `codeql` (gratuita per progetti open source) per eseguire analisi avanzate su Python e JavaScript/TypeScript.
14. **Check di conformità alle policy**: integrare script che verificano l'aderenza alle policy di sicurezza (es. assenza di dati personali nei log, uso di protocolli cifrati).

15. **Pipeline di sicurezza dedicata**: creare un workflow `security.yml` che si attivi su ogni push/PR per eseguire tutti i controlli sopra. Questo workflow potrà fallire la build se vengono rilevate vulnerabilità gravi.

16. **Protezione dei dati e privacy**

17. **Minimizzazione dei dati**: assicurarsi che i file di telemetria raccolgano solo dati necessari al game design.
18. **Pseudonimizzazione**: implementare meccanismi per anonimizzare gli ID dei giocatori nei log e nei report.
19. **Gestione consensi**: documentare (nel GDD e nell'app) come viene richiesto e registrato il consenso al trattamento dei dati.
20. **Crittografia in transito e a riposo**: utilizzare HTTPS per le API e crittografare i database dove sono conservate statistiche e telemetria.

21. **Piano di retention**: definire la durata di conservazione dei dati di telemetria e la procedura per la loro cancellazione automatica.

22. **Formazione e cultura della sicurezza**

23. Organizzare sessioni interne per formare il team su secure coding, OWASP Top 10, gestione dei segreti e privacy.
24. Predisporre un canale dedicato (es. chat o board del progetto) per comunicare vulnerabilità, aggiornamenti di sicurezza e best practice.
25. Prevedere code review con checklist di sicurezza (Injection, XSS, gestione errori, logging sicuro).

26. Incorporare la verifica della sicurezza nei criteri di accettazione delle user stories.

27. **Piano di risposta agli incidenti**

28. Definire una procedura da seguire in caso di violazione della sicurezza: rilevamento, contenimento, analisi, recupero e comunicazione.
29. Preparare un documento di contingenza e un team responsabile con ruoli chiari.
30. Eseguire esercitazioni periodiche per testare la prontezza del team.

Prossimi passi

- Creare una directory `docs/security/` nel repository (da realizzare tramite commit) in cui collocare:
- la **politica di sicurezza e privacy**,
- il **threat model** (diagramma e descrizione delle superfici di attacco),
- il **registro dei rischi** con stato aggiornato,
- il **piano di risposta agli incidenti**.
- Implementare il workflow `security.yml` in `.github/workflows/` per automatizzare i controlli (SAST, dependency auditing, secret scanning).
- Configurare e abilitare GitHub Dependabot, secret scanning e Code QL dalle impostazioni del repository.
- Aggiornare la documentazione dell'onboarding per includere la formazione sulla sicurezza.

Conclusioni

Integrare la sicurezza nel ciclo di vita del progetto Evo-Tactics non è un compito una tantum ma un processo continuo. Creando politiche chiare, effettuando threat modeling e risk assessment regolari e automatizzando i controlli nelle pipeline, è possibile ridurre drasticamente la superficie d'attacco e aumentare la fiducia degli utenti. Seguendo le linee guida del Secure SDLC [1], il team potrà prevenire vulnerabilità e proteggere i dati dei giocatori fin dalle prime fasi di sviluppo.

---

[1] (Microsoft Word - Allegato 1- Linee Guida per l\222adozione di un ciclo di sviluppo di software sicuro)
https://www.agid.gov.it/sites/agid/files/2024-06/Linee_guida_per_ladozione_di_un_ciclo_di_sviluppo_di_software_sicuro.pdf