Proyecto Final	**Secure Software Development Life Cycle (SSDLC)**
Fecha de entrega	**Sábado 15 de noviembre 2024 23:59:59**

Objetivo

- El objetivo es que se conformen equipos de 4 a 5 miembros máximo, donde cada miembro se asegure de contribuir en una de las distintas fases del ciclo de vida del proyecto llamado UNA-chat, aplicando cualquiera de los controles de seguridad explicados abajo. Tomar en cuenta que las contribuciones se evaluaran de manera individual.

Indices

Desarrollo

1. ### Implementar escáner estático local (VS Code Extension)
    - **Definir un estándar de desarrollo y convenciones o estilos a seguir por cada miembro del equipo para asegurar siempre en lo posible el mismo nivel de calidad en cada desarrollo. Incluir para esto reglas para agentes de IA, que sigan buenas prácticas de seguridad, forzar buenas prácticas como usar BDD, siempre pasar linting, siempre hacer build del projecto, entre otras, establecer convenciones de código, entre otras.**

    - **Levantar la lista de Materiales del Software (SBOM)**

        - Construir una lista SBOM para la aplicación, incluir: Bibliotecas de terceros, Módulos, Frameworks, Dependencias, Datos sobre las versiones, Licencias asociadas, y el riesgo inherente que estas librerías presentan en una matriz de riesgos…

        - Ejemplo. La librería Patitos.js representa un riesgo algo de ser vulnerable por que su repositorio no se actualiza desde hace tantos años y el mantenimiento solo se lo da una persona. Su riesgo de presentar brechas de seguridad que afecten la disponibilidad del sistema es de Probabilidad alta 4 de 5, e Impacto bajo 2 de 5. Por lo tanto su riesgo inherente es 8. Lo cual está por encima del umbral de tolerancia del riesgo. Se recomienda prevenir usar la librería y en su lugar recomendamos esta otra.

    - **VS Code Extensions**

- En VS Code, integrar herramientas para escaner el código detectar vulnerabilidades de seguridad, code smells, a nivel del IDE.
- Ejemplos:
- Semgrep:

- Snyk
- Instalar la extensión de Snyk o Semgrep y ponerla a prueba en una Demo.

- **Linting**:
  - En VS Code, integrar herramientas para analizar el código y detectar errores sintácticos, “**typos**”, y cualquier otra que ayude a prevenir romper las convenciones o el estándar de desarrollo del equipo.

2. Implementar escáner estático (SAST) + Linting + UnitTest en GitHub Actions.

- **SAST**
  - En el pipeline de GitHub Actions debe integrar un escaner de código estático como Snyk, SonnarQ, (investigar alternativas) para buscar y detectar vulnerabilidades de seguridad, code smells, etc.
- **Unit Test**
  - Debe asegurarse que el projecto cuente con Unit Test adecuado basado en BDD, el pipeline ejecutará las pruebas unitarias y que no proceda a un siguiente paso hasta que estas pruebas pasen por completo.
- **Linting**:
  - En el pipeline de GitHub Actions debe integrar un analizador de código para detectar errores sintácticos y de forma que puedan romper las convenciones o el estándar de desarrollo del equipo.

3. Escaneo de imágenes de contenedores antes del despliegue

- Debe asegurarse de que la solución es conteinerizada. Ayudar a que el pipeline en GitHub Actions build y haga push de la imagen al repositorio de imágenes en DockerHub con la etiqueta development. Debe crear un Job para garantizar que las imágenes de Docker utilizadas en el proyecto estén libres de vulnerabilidades conocidas, para esto se recomienda investigar e implementar herramientas de **escaneo estático de vulnerabilidades en contenedores** (como Clair o Anchore) que se pueden integrar dentro de flujos de **GitHub Action.**

4. Testing y Monitoreo de seguridades – Bug Bounty

- Despliegue de la infraestructura de development:
  - Una vez que el código se fusiona en una (ej. Development), se despliega automáticamente el Backend in Frontent del projecto a un ambiente Cloud como Azure, AWS, o alguna plataforma (PaaS) como render, netlify, vercel, o similar.

- Una vez desplegado el proyecto deben hacerse las pruebas de Nuclei en el Pipeline CI/CD:

  - Nuclei: https://blog.projectdiscovery.io/implementing-nuclei-into-your-github-ci-cd-for-scanning-live-web-applications/

  - Nuclei Action: https://github.com/projectdiscovery/nuclei-action?ref=blog.projectdiscovery.io

Aparte de esto debe realizarse una integración correcta de monitoreo con Logs, Metrics, Traces y alertas. El sistema debe ser capaz de avisar si está frente a un ataque o a un pico de tráfico.

El objetivo de este punto en todo caso es tener herramientas de seguridad ofensiva y defensiva posterior al despliegue de la aplicación.

Para aclarar este punto.
Seguridad Ofensiva: busca vulnerabilidades atacando (conscientemente, autorizado).

Seguridad Defensiva: protege, monitoriza y responde ante ataques (firewalls, SIEM, monitoreo, hardening).

Presentación Final:

Al final, deberán presentarse los resultados del trabajo en una exposición tipo (DEMO) al grupo, justificando cómo cada implementación contribuye a la seguridad del proyecto en cada etapa del SSDLC, retos encontrados, recomendaciones. Se debe dar evidencia clara del pipeline funcional, resultados del escaneo en cada etapa, y lograr convencer a la gerencia (profesor) porque estas soluciones deben verse más que como un gasto, como una inversión.

Entregables

1. **Investigación y Desarrollo:**
    a. **Repositorio en GitHub:** El proyecto debe estar alojado en un repositorio de GitHub publico donde se pueda verificar la configuración del pipeline y los escáneres implementados, compartir enlace del repositorio en profelink junto con el enlace de médium + la presentación final (PPT o Canva esta es opcional) en un .ZIP **5%**
    b. **Documentación:** Un breve tutorial en **Medium**, donde se expliquen los pasos para que cualquier equipo pueda configurar su ambiente de desarrollo de la misma forma en pasos sencillos. **10%**
2. **Exposición y defensa del proyecto: 15%**

Recursos:

SAST – Static Applicaction Security Testing

Herramientas de revisión estática de código que trabajan con el código fuente y buscan patrones conocidos y relaciones entre métodos, variables, clases y bibliotecas. SAST trabaja con el código en bruto y, por lo general, no con paquetes compilados.

Name	URL	Description	Meta
Brakeman	https://github.com/presidentbeef/brakeman	Brakeman is a static analysis tool which checks Ruby on Rails applications for security vulnerabilities	STARS 6.8K
Semgrep	https://semgrep.dev/	Hi-Quality Open source, works on 17+ languages	STARS 9.3K

Bandit	https://github.com/PyCQA/bandit	Python specific SAST tool	STARS 5.8K
libsast	https://github.com/ajinabraham/libsast	Generic SAST for Security Engineers. Powered by regex-based pattern matcher and semantic-aware Semgrep	STARS 112
ESLint	https://eslint.org/	Find and fix problems in your JavaScript code	
nodejsscan	https://github.com/ajinabraham/nodejsscan	NodeJs SAST scanner with GUI	STARS 2.3K
FindSecurityBugs	https://find-sec-bugs.github.io/	The SpotBugs plugin for security audits of Java web applications	STARS 2.2K

SonarQube Community	https://github.com/SonarSource/sonarqube	Detect security issues in code review with Static Application Security Testing (SAST)	STARS 8.4K
gosec	https://github.com/securego/gosec	Inspects source code for security problems by scanning the Go AST.	STARS 7.3K
Safety	https://github.com/pyupio/safety	Checks Python dependencies for known security vulnerabilities.	STARS 1.6K

Nota:

Semgrep es una herramienta CLI gratuita; sin embargo, algunos conjuntos de reglas tienen diferentes licencias, que incluyen tanto opciones gratuitas como comerciales. Lista curada de herramientas SAST por OWASP: https://owasp.org/www-community/Source_Code_Analysis_Tools

Container Security Tools

Name	URL	Description	Meta
Harbor	https://github.com/goharbor/harbor	Trusted cloud-native registry project	STARS 22K
Anchore	https://github.com/anchore/anchore-engine	Centralized service for inspection, analysis, and certification of container images	STARS 1.6K
Clair	https://github.com/quay/clair	Docker vulnerability scanner	STARS 22K
Deepfence ThreatMapper	https://github.com/deepfence/ThreatMapper	Apache v2, powerful runtime vulnerability scanner for Kubernetes, virtual machines,	STARS 4.5K

		and serverless environments	
Docker bench	https://github.com/docker/docker-bench-security	Docker benchmarking against CIS	STARS 22K
Falco	https://github.com/falcosecurity/falco	Container runtime protection	STARS 6.6K
Trivy	https://github.com/aquasecurity/trivy	Comprehensive scanner for vulnerabilities in container images	STARS 20K
Notary	https://github.com/notaryproject/notary	Docker signing	STARS 3.1K
Cosign	https://github.com/sigstore/cosign	Container signing	STARS 3.9K
Watchtower	https://github.com/containrrr/watchtower	Updates the running version of your containerized app	STARS 16K

Grype	https://github.com/anchore/grype	Vulnerability scanner for container images (and also filesystems)	

Referencias

List of resources worth investigating:
https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf

https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsStrategyGuide.pdf

https://csrc.nist.gov/publications/detail/sp/800-204c/draft

https://owasp.org/www-project-devsecops-maturity-model/

https://www.sans.org/posters/cloud-security-devsecops-best-practices/