Desafío de tripulaciones: Equipo 2
Vertical: Ciberseguridad Septiembre 24 FT

www.felgtbi.org

FELGTBI+
Federación Estatal de Lesbianas, Gais, Trans, Bisexuales, Intersexuales y más

Informe técnico de Ciberseguridad

Equipo:
Marco Pérez
Francisco Herrera
Alfonso Cascajero
Jaime Fernández
Andrés Ramos
Pelayo Hernández

Índice

Introducción al Desafío de Tripulaciones

En el Desafío de Tripulaciones, colaboramos como estudiantes de Data Science, Ciberseguridad y Full Stack para aplicar nuestros conocimientos en un proyecto real, diseñado en conjunto con un socio estratégico.

En esta edición, trabajamos con La Federación Estatal de Lesbianas, Gais, Trans, Bisexuales, Intersexuales y más, que es una ONG estatal de carácter laico, laicista, feminista, apartidista y asindicalista que agrupa a más de 50 entidades LGTBI+ de todo el territorio español. Trabajan por la diversidad afectivo-sexual, familiar y de género, así como por la lucha contra el vih y sida.

Nuestro Reto

Nuestro desafío se enfoca en crear un chat bot intuitivo y accesible que ofrezca atención y orientación sobre el vih con el objetivo de proporcionar información precisa, reducir el estigma y ofrecer apoyo. Dirigido a personas con diagnóstico de vih o que creen que pueden haber contraído vih. También para personal sociosanitario para que pueda orientar a las personas usuarias.

Enfoque en Ciberseguridad

En este informe, nos centraremos en asegurar la aplicación desde el punto de vista de la ciberseguridad. Abordaremos los siguientes aspectos con la ayuda de herramientas y metodologías especializadas:

- **Entender el Contexto Global:** Utilizaremos **Microsoft Threat Modeling Tool** para analizar el contexto funcional de la aplicación, identificando características sensibles y posibles vectores de ataque.

- **Hardening:** Realizaremos el hardening de sistemas operativos y aplicaciones necesarias para el proyecto, siguiendo guías de hardening.

- **Infraestructura:** Propondremos una infraestructura que sea jerarquizable, modular y ampliable para el despliegue futuro de la aplicación, utilizando la Microsoft Threat Modeling Tool para garantizar que cumple con los requisitos de seguridad desde el diseño.

- **Backup:** Implementaremos un sistema de backup con la regla del 3:2:1, considerando la estrategia, costes, periodos de conservación y la sensibilidad de los datos almacenados.

- **OWASP Top 10:** Utilizaremos la metodología del OWASP Top 10 para identificar agentes maliciosos y vectores de ataque, y para definir controles de seguridad adecuados.

● **Controles de Seguridad:** Evaluaremos los controles de seguridad detenidos durante la etapa de diseño para asegurar que cumplen con las buenas prácticas del SSDLC **(Secure Software Development Life Cycle).**

● **Pruebas de Código Estático:** Ejecutaremos pruebas con herramientas como **SonarQube** para detectar vulnerabilidades y malas prácticas durante el desarrollo.

● **Auditoría de Pentesting:** Al analizar el desarrollo, realizaremos una auditoría rápida con herramientas de pentesting para identificar problemas evidentes sin entrar en el detalle de una auditoría exhaustiva.

● **Evaluación de Cumplimiento Normativo:** Realizaremos una evaluación para asegurar que la aplicación cumple con las normativas y regulaciones relevantes de protección de datos y privacidad, como el GDPR o la CCPA, dependiendo del contexto y ubicación de los datos.

Durante las últimas semanas del Bootcamp, aplicamos nuestras habilidades en ciberseguridad y colaboramos con las otras verticales para desarrollar una solución integral y segura.

Impacto técnico y de negocio

1. Impacto Técnico:

- Se refiere a cómo el diseño, la implementación y las operaciones del sistema afectan los aspectos técnicos de la organización. Esto incluye:
  - **Integración**: Compatibilidad con sistemas existentes.
  - **Seguridad**: Cumplimiento de estándares y reducción de riesgos técnicos.
  - **Desempeño**: Eficiencia, escalabilidad y sostenibilidad técnica.
  - **Mantenibilidad**: Facilidad de actualización y soporte.
  - **Confiabilidad**: Resiliencia y disponibilidad del sistema.

2. Impacto de Negocio:

- Se centra en los beneficios o riesgos que el proyecto aporta a los objetivos estratégicos y operativos de la organización. Esto incluye:
  - **Cumplimiento**: Adherencia a normativas legales o regulatorias.

- **Rentabilidad**: Impacto en los ingresos, costos y retorno de inversión (ROI).
- **Experiencia del cliente**: Mejoras en la usabilidad o satisfacción.
- **Riesgos de reputación**: Cómo los fallos podrían dañar la percepción de la marca.
- **Eficiencia operativa**: Optimización de procesos de negocio.

Evaluar si el diseño sigue las buenas prácticas del SSDLC

El **Secure Software Development Life Cycle (SSDLC)** integra buenas prácticas de seguridad en cada fase del ciclo de vida del desarrollo. Para determinar si un proyecto sigue estas prácticas, se deben realizar los siguientes pasos:

1. Revisión de Requisitos:

- Verificar que los requisitos de seguridad están definidos claramente.
- Evaluar si se han considerado amenazas y riesgos desde el inicio (por ejemplo, análisis de amenazas).

2. Diseño Seguro:

- Confirmar que el diseño incluye medidas de seguridad específicas, como autenticación, autorización, encriptación y manejo seguro de errores.
- Evaluar el uso de modelos como STRIDE o DREAD para identificar y mitigar riesgos.

3. Codificación Segura:

- Asegurarse de que los desarrolladores sigan guías de codificación segura (por ejemplo, OWASP Top 10).
- Revisar el uso de herramientas de análisis estático y dinámico para identificar vulnerabilidades.

4. Pruebas de Seguridad:

- Verificar si se realizan pruebas específicas, como pruebas de penetración, fuzzing y validación de entradas.
- Confirmar la existencia de un plan para abordar vulnerabilidades encontradas.

5. Implementación y Mantenimiento:

- Validar que el proceso de despliegue incluye revisiones de configuración y pruebas finales de seguridad.

- Revisar si hay un plan de mantenimiento para actualizaciones y parches regulares.

6. Monitoreo y Respuesta:

- Asegurarse de que el sistema incluye capacidades de monitoreo para detectar incidentes de seguridad en tiempo real.
- Verificar que exista un plan de respuesta a incidentes.

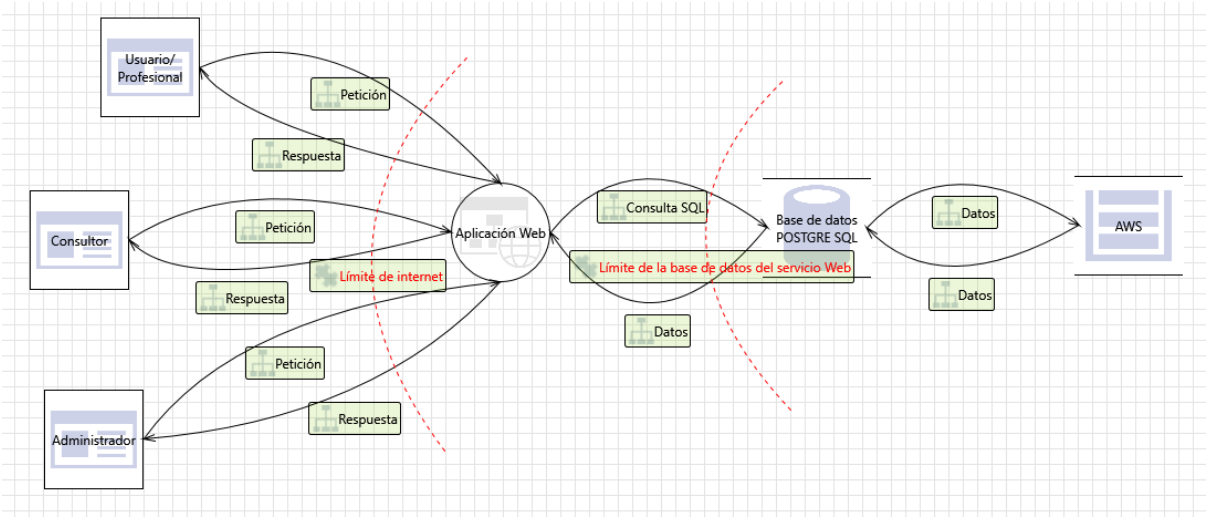
Resultado Final: Si se identifican deficiencias en alguna fase, se debe priorizar su resolución para cumplir con las mejores prácticas del SSDLC.

Threat Modeling Tool

Microsoft Threat Modeling Tool es una herramienta esencial para identificar y analizar amenazas en sistemas y aplicaciones. Facilita la creación de diagramas de arquitectura, la evaluación de posibles amenazas y la elaboración de estrategias de mitigación.

En nuestro trabajo, utilizamos Microsoft Threat Modeling Tool para comprender a fondo el contexto global de la aplicación que estamos desarrollando y la infraestructura de red en la que operará. El proceso comienza con la creación de un modelo detallado de la arquitectura de la aplicación. Este modelo nos permite visualizar cómo interactúan los distintos componentes y cómo se comunican entre sí, ayudándonos a identificar posibles puntos débiles y a comprender el flujo de datos.

A continuación, examinamos la infraestructura de red, incluyendo servidores, nubes de almacenamiento, bases de datos, usuarios y cualquier otro dispositivo que forme parte del entorno en el que la aplicación se desplegará. Evaluamos cómo se conectan estos componentes y cómo se protegen las comunicaciones entre ellos. Esto nos permite anticipar posibles amenazas externas y problemas de configuración que podrían afectar la seguridad y el rendimiento de la aplicación.



Con una visión completa del contexto global y de la infraestructura, podemos planificar mejor las estrategias de mitigación de riesgos, implementar controles de seguridad adecuados y garantizar que nuestra aplicación sea robusta y segura en el entorno en el que operará.

Una vez desarrollado el modelo, tenemos una visión clara de dónde podríamos colocar firewalls y otros dispositivos de seguridad para proteger nuestra infraestructura. Identificamos los puntos críticos que necesitan ser reforzados y establecemos medidas de protección adecuadas. Además, sabemos dónde debemos hacer un esfuerzo adicional en la hardenización, como en la configuración de sistemas y en la aplicación de políticas de seguridad estrictas.

Por ejemplo, hemos identificado que la conexión de candidatos al servidor es un punto vulnerable que requiere especial atención. Asegurarnos de que esta conexión esté bien protegida y controlada es esencial para evitar posibles ataques y garantizar la seguridad general del sistema.

1. **Acceso no autorizado a la base de datos debido a reglas de autorización laxas.**

Descripción.

Aplicar principio de mínimos privilegios. Los usuarios deben tener únicamente permiso para realizar su tarea, ninguna otra.

Posibles mitigaciones.

El usuario que realiza la consulta SQL entre aplicación web y Base de datos POSTGRE SQL (www-data) debe tener permisos únicamente de lectura.

Referencia: https://aka.ms/tmtauthz#privileged-server

2. **Acceso a datos confidenciales al rastrear el tráfico a la base de datos.**

Descripción.

Se pueden espiar las comunicaciones entre el servidor de aplicaciones y el servidor de la base de datos debido al uso del protocolo de comunicación de texto sin cifrar.

Posibles mitigaciones.

Asegurarse de que la conexión del servidor SQL esté cifrada y valide el certificado. Los pasos se encuentran en la referencia

Referencia: https://learn.microsoft.com/es-es/sql/database-engine/configure-windows/configure-sql-server-encryption?view=sql-server-ver16

3. **Inyecciones SQL dinámicas.**

Descripción.

Es imprescindible que no se puedan realizar inyecciones SQL en ningún cajetín.

Posibles mitigaciones.

En la referencia hay un ejemplo de una práctica incorrecta y la formación del procedimiento de manera segura. La principal diferencia es el uso del procedimiento almacenado uspGetProductsByCriteriaSecure en vez de uspGetProductsByCriteria

Referencia: https://aka.ms/tmtinputval#stored-proc

4. **Divulgación de información.**

Descripción.

Es imprescindible que no se expongan detalles de seguridad en mensajes de error.

Posibles mitigaciones.

Evitar que se muestren datos como: Procedimientos SQL, detalles de errores SQL dinámicos, seguimiento de la pila, valores de la configuración…

Referencia: https://aka.ms/tmtxmgmt#messages

5. **Un adversario puede falsificar la aplicación web de destino debido a una configuración insegura del certificado TLS.**

Descripción.

¿Se utilizan certificados para la comunicación?

Posibles mitigaciones.

Si es así comprobar los puntos recogidos en la referencia.

6. **Posible robo de datos confidenciales como credenciales de usuario.**

Posibles mitigaciones.

Deshabilite explícitamente el atributo de autocompletar en formularios y entradas sensibles https://aka.ms/tmtdata#autocomplete-input.

Realice validación y filtro de entradas en todas las propiedades del modelo de tipo cadena https://aka.ms/tmtinputval#typemodel.

Valide que todas las redirecciones dentro de la aplicación estén cerradas o se realicen de forma segura https://aka.ms/tmtinputval#redirect-safe.

Implemente funciones de contraseña olvidada de forma segura:

https://aka.ms/tmtauthn#forgot-pword-fxn.

7. **Un adversario puede crear un sitio falso y lanzar ataques de phishing.**

Categoría.

Suplantación

Descripción.

El phishing busca obtener información sensible (como nombres de usuario, contraseñas y detalles de tarjetas de crédito), haciéndose pasar por un servidor web confiable.

Posibles mitigaciones.

Verificar certificados X.509 para autenticar conexiones SSL, TLS y DTLS.

Incorporar defensas contra UI redressing o clickjacking en páginas ASP.NET autenticadas.

Validar que todos los redireccionamientos estén cerrados o se realicen de manera segura.

Referencia:

https://aka.ms/tmtcommsec#x509-ssltl

8. **Un adversario puede suplantar la base de datos POSTGRE SQL y obtener acceso a la aplicación web.**

    Descripción.

Si no se implementa una autenticación adecuada, un adversario puede suplantar un proceso de origen o una entidad externa y obtener acceso no autorizado a la aplicación web.

Posibles mitigaciones.

Considere usar un mecanismo de autenticación estándar para autenticar el acceso a la aplicación web.

Consulte:

https://aka.ms/tmtauthn#standard-authn-web-app

9. **Un adversario puede obtener acceso a datos sensibles realizando inyección SQL a través de la aplicación web.**

    Descripción.

La inyección SQL es un ataque en el que se inserta código malicioso en cadenas que posteriormente se pasan a una instancia de SQL Server para su análisis y ejecución. La forma principal de inyección SQL consiste en la inserción directa de código en variables de entrada proporcionadas por el usuario, que luego se concatenan con comandos SQL y se ejecutan. Un ataque menos directo inyecta código malicioso en cadenas destinadas a almacenarse en una tabla o como metadatos. Cuando estas cadenas almacenadas se concatenan posteriormente en un comando SQL dinámico, se ejecuta el código malicioso.

Posibles mitigaciones.

Asegúrese de que se utilicen parámetros con tipo seguro (type-safe) en la aplicación web para el acceso a datos. Consulte: https://aka.ms/tmtinputval#typesafe

10. **Un adversario puede obtener acceso a datos sensibles almacenados en los archivos de configuración de la aplicación web.**

    Descripción.

Un adversario puede obtener acceso a los archivos de configuración, y si contienen datos sensibles, estos quedarían comprometidos.

Posibles mitigaciones.

Cifre las secciones de los archivos de configuración de la aplicación web que contengan datos sensibles. Consulte: https://aka.ms/tmtdata#encrypt-data

11. **Un adversario puede obtener acceso no autorizado a la base de datos debido a la falta de protección de acceso a la red.**

Descripción.

Si no hay restricciones a nivel de red o del firewall del host para acceder a la base de datos, cualquiera podría intentar conectarse a esta desde una ubicación no autorizada.

Posibles mitigaciones.

Configure un firewall de Windows para el acceso al motor de base de datos.

Consulte:

https://aka.ms/tmtconfigmgmt#firewall-db

12. **Un adversario puede obtener acceso no autorizado a la base de datos debido a reglas de autorización laxas.**

Descripción.

El acceso a la base de datos debe configurarse con roles y privilegios basados en el principio de privilegio mínimo y necesidad de saber.

Posibles mitigaciones.

Asegúrese de que se utilicen cuentas con privilegios mínimos para conectarse al servidor de base de datos.

Consulte:

https://aka.ms/tmtauthz#privileged-server

Implemente seguridad a nivel de filas (RLS) para evitar que los inquilinos accedan a los datos de otros.

Consulte:

https://aka.ms/tmtauthz#rls-tenants

El rol de administrador del sistema (Sysadmin) solo debe incluir usuarios válidos y necesarios.

Consulte:

https://aka.ms/tmtauthz#sysadmin-users

13. **Un adversario puede acceder a datos sensibles realizando una inyección SQL.**

Descripción.

La inyección SQL es un ataque en el cual se inserta código malicioso en cadenas que luego se pasan a una instancia de SQL Server para su análisis y ejecución. La forma principal de inyección SQL consiste en la inserción directa de código en variables de entrada de usuario que se concatenan con comandos SQL y se ejecutan. Un ataque menos directo inyecta código malicioso en cadenas destinadas a ser almacenadas en una tabla o como metadatos. Cuando estas cadenas almacenadas se concatenan posteriormente en un comando SQL dinámico, el código malicioso se ejecuta.

Posibles mitigaciones.

Asegúrese de que la auditoría de inicio de sesión esté habilitada en el servidor SQL.

Consulte.
https://aka.ms/tmtauditlog#identify-sensitive-entities

Garantice el uso de cuentas con privilegios mínimos para conectarse al servidor de base de datos.

Consulte.
https://aka.ms/tmtauthz#privileged-server

Habilite la detección de amenazas en la base de datos SQL de Azure.

Consulte.
https://aka.ms/tmtauditlog#threat-detection

Evite el uso de consultas dinámicas en procedimientos almacenados.

Consulte.
https://aka.ms/tmtinputval#stored-proc

14. **Un adversario puede negar acciones en la base de datos debido a la falta de auditoría.**

Descripción.

El registro adecuado de todos los eventos de seguridad y las acciones de los usuarios genera trazabilidad en un sistema y niega posibles problemas de repudio. En ausencia de controles

adecuados de auditoría y registro, sería imposible implementar cualquier tipo de responsabilidad en el sistema.

Posibles mitigaciones.

Asegúrese de que la auditoría de inicio de sesión esté habilitada en el servidor SQL.

Consulte.
https://aka.ms/tmtauditlog#identify-sensitive-entities

15. **Un adversario puede aprovechar la falta de sistemas de monitoreo y generar tráfico anómalo hacia la base de datos.**

Descripción.

Un adversario puede aprovechar la falta de detección y prevención de intrusiones ante actividades anómalas en la base de datos y generar tráfico anómalo hacia la base de datos.

Posibles mitigaciones.

Habilitar la detección de amenazas en la base de datos SQL de Azure.

Consulte.

https://aka.ms/tmtauditlog#threat-detection

16. **Un adversario puede realizar acciones en nombre de otro usuario debido a la falta de controles contra solicitudes entre dominios.**

Descripción.

No restringir las solicitudes que se originan en dominios de terceros puede dar lugar a acciones o acceso no autorizados a los datos.

Posibles mitigaciones.

Asegúrese de que las páginas ASP.NET autenticadas incorporen protección de interfaz de usuario o defensas contra el secuestro de clics.

Consulte.

https://aka.ms/tmtconfigmgmt#ui-defenses

Asegúrese de que solo se permiten orígenes confiables si CORS está habilitado en aplicaciones web ASP.NET.

Consulte.

https://aka.ms/tmtconfigmgmt#cors-aspne

Mitigar los ataques de falsificación de solicitudes entre sitios (CSRF) en páginas web ASP.NET.

Consulte.

https://aka.ms/tmtsmgmt#csrf-asp

17. **Un adversario puede obtener acceso a datos confidenciales de archivos de registro.**

    Descripción.

Un adversario puede obtener acceso a datos confidenciales de archivos de registro.

Posibles mitigaciones.

Asegúrese de que la aplicación no registre datos confidenciales del usuario.

Consulte

https://aka.ms/tmtauditlog#log-SENSITIVE-data

Asegúrese de que los archivos de auditoría y de registro tengan Acceso restringido.

Consulte.

https://aka.ms/tmtauditlog#log-restricted-access

18. **Un adversario puede obtener acceso a determinadas páginas o al sitio en su conjunto.**

    Descripción.

Robots.txt se encuentra a menudo en el directorio raíz de su sitio y existe para regular los robots que rastrean su sitio. Aquí es donde puede otorgar o denegar permiso a todos o algunos robots de motores de búsqueda específicos para acceder a determinadas páginas o a su sitio en su conjunto. El estándar para este archivo se desarrolló en 1994 y se conoce como Estándar de exclusión de robots o Protocolo de exclusión de robots. Puede encontrar información detallada sobre el protocolo robots.txt en robotstxt.org.

Posibles mitigaciones.

Asegúrese de que las interfaces administrativas estén bloqueadas adecuadamente.

Consulte.

https://aka.ms/tmtauthn#admin-interface-lockdown

19. **Un adversario puede crear un sitio web falso y lanzar ataques de phishing.**

    Descripción.

El phishing intenta obtener información confidencial como nombres de usuario, contraseñas y detalles de tarjetas de crédito (y a veces, indirectamente, dinero), a menudo por razones maliciosas, haciéndose pasar por un servidor web que es una entidad confiable en las comunicaciones electrónicas.

Posibles mitigaciones.

Verifique los certificados X.509 utilizados para autenticar conexiones SSL, TLS y DTLS.

Consulte. https://aka.ms/tmtcommsec#x509-ssltls

Asegúrese de que las páginas ASP.NET autenticadas incorporen UI Redressing o defensas contra el clickjacking.

Consulte.

https://aka.ms/tmtconfigmgmt#ui-defenses

Valide que todas las redirecciones dentro de la aplicación estén cerradas o realizadas de forma segura .

Consulte

https://aka.ms/tmtinputval#redirect-safe

20. **Un adversario puede dañar la aplicación web de destino inyectando código malicioso o cargando archivos peligrosos.**

    Description.

La desfiguración de un sitio web es un ataque a un sitio web en el que el atacante cambia la apariencia visual del sitio o de una página web.

Posibles mitigaciones.

Implementar la política de seguridad de contenido (CSP) y deshabilitar JavaScript en línea.

Consulte.

https://aka.ms/tmtconfigmgmt#csp-js

Habilite el filtro XSS del navegador.

Consulte.

https://aka.ms/tmtconfigmgmt#xss-filter

Acceda a javascripts de terceros únicamente desde fuentes confiables.

Consulte.

https://aka.ms/tmtconfigmgmt#js-trusted

21. **Un adversario puede obtener acceso a datos confidenciales realizando una inyección de SQL a través de la aplicación web.**

    Descripción.

La inyección SQL es un ataque en el que se inserta código malicioso en cadenas que luego se pasan a una instancia de SQL Server para su análisis y ejecución. La forma principal de inyección SQL consiste en la inserción directa de código en variables de entrada del usuario que se concatenan con comandos SQL y se ejecutan. Un ataque menos directo inyecta código malicioso en cadenas destinadas al almacenamiento en una tabla o como metadatos. Cuando las cadenas almacenadas se concatenan posteriormente en un comando SQL dinámico, se ejecuta el código malicioso.

Posibles mitigaciones.

Asegúrese de que se utilicen parámetros de tipo seguro en la aplicación web para el acceso a los datos.

Consulte:

https://aka.ms/tmtinputval#typesafe

22. **Un adversario puede obtener acceso a datos confidenciales almacenados en los archivos de configuración de la aplicación web.**

    Descripción.

Un adversario puede obtener acceso a los archivos de configuración. y si en él se almacenan datos confidenciales, se verían comprometidos.

Posibles mitigaciones.

Cifrar secciones de los archivos de configuración de la aplicación web que contienen datos confidenciales.

Consulte:

https://aka.ms/tmtdata#encrypt-data

Hardening

1. Hardenización del Servidor

1.1 Deshabilitar Servicios Innecesarios

Desactiva cualquier servicio que no sea esencial para el funcionamiento del chatbot. Esto reduce la superficie de ataque.

1. Identificar servicios activos:

- Ejecuta systemctl list-units --type=service o chkconfig --list para listar los servicios activos en el servidor.

2. Deshabilitar servicios no esenciales:

- Usa systemctl disable <nombre del servicio> o chkconfig <nombre del servicio> off.

3. Verificar cambios:

- Reinicia el servidor y confirma que solo los servicios necesarios están en ejecución.

1.2 Actualizaciones Regulares

Mantén el sistema operativo y el software del servidor actualizados con los últimos parches de seguridad. Usa AWS Systems Manager para automatizar este proceso.

Configura AWS Systems Manager para automatizar actualizaciones:

- Habilita Patch Manager en Systems Manager.
- Define una "baseline" para aplicar solo los parches aprobados.

Verifica que el servidor esté actualizado:

- Ejecuta yum update (Amazon Linux) o apt update && apt upgrade (Ubuntu).

1.3 Contraseñas Seguras

Utiliza contraseñas complejas y únicas para todas las cuentas de usuario. Implementa políticas de rotación de contraseñas.

1. Define una política de contraseñas:

- Edita /etc/login.defs para establecer requisitos mínimos como longitud y complejidad.

2. Fuerza la rotación de contraseñas:
    - Configura PASS_MAX_DAYS en /etc/login.defs.
3. Revisa y elimina contraseñas predeterminadas:
    - Ejecuta passwd -l <usuario> para bloquear cuentas no utilizadas.

2. Hardenización de la Red

2.1 Configuración de Firewalls

Configura los grupos de seguridad de AWS para restringir el tráfico entrante y saliente solo a los puertos y servicios necesarios. Utiliza Network ACLs para una capa adicional de seguridad.

1. Configura Grupos de Seguridad (SG):
    - Permite solo los puertos esenciales, como 443 (HTTPS) y 22 (SSH).
    - Deniega todo el tráfico entrante por defecto.
2. Configura Network ACLs (NACL):
    - Implementa reglas adicionales para controlar el tráfico en subredes de Amazon VPC.
3. Verifica configuraciones:
    - Usa aws ec2 describe-security-groups y aws ec2 describe-network-acls para revisar.

2.2 Cifrado del Tráfico

Asegura que todo el tráfico de red esté cifrado utilizando TLS/SSL. Configura Amazon CloudFront y AWS Certificate Manager para gestionar certificados SSL.

1. Genera certificados TLS/SSL con AWS Certificate Manager (ACM):
    - Asocia los certificados con un ALB (Application Load Balancer).
2. Configura Amazon CloudFront:
    - Habilita HTTPS obligatorio.
    - Fuerza el uso de protocolos modernos (TLS 1.2 o superior).

2.3 Segmentación de la Red

Utiliza Amazon VPC para crear subredes privadas y públicas. Segmenta los recursos según su función y sensibilidad. Implementa VPC Flow Logs para monitorear el tráfico de red.

1. Crea subredes públicas y privadas:
    - Asegúrate de que los servidores críticos están en subredes privadas.
2. Configura rutas apropiadas:
    - Asegúrate de que solo los recursos en subredes públicas tengan acceso a internet.
3. Activa VPC Flow Logs:
    - Configúlo para monitorear todo el tráfico en la red.

3. Hardenización de Aplicaciones

3.1 Contraseñas Predeterminadas

Cambia todas las contraseñas predeterminadas de las aplicaciones y servicios. Esto incluye bases de datos, servidores web y cualquier otro componente.

1. Cambia las contraseñas predeterminadas de aplicaciones y servicios.
    - Para bases de datos: Actualiza contraseñas usando ALTER USER en PostgreSQL.
2. Deshabilita cuentas por defecto no necesarias.

3.2 Eliminación de Componentes Innecesarios

Elimina cualquier componente o función que no sea necesaria para el funcionamiento del chatbot. Esto reduce la superficie de ataque.

1. Revisa los servicios y bibliotecas en uso:
    - Usa pip list para Python o npm list para Node.js.
2. Elimina dependencias obsoletas o sin uso.

3.3 Inspección de Integraciones

Revisa las integraciones con otras aplicaciones y sistemas para eliminar privilegios innecesarios. Asegúrate de que las integraciones sean seguras y estén actualizadas.

1. Evalúa las integraciones actuales:
    - Usa AWS Identity and Access Management (IAM) para revisar permisos.
2. Limita los permisos:
    - Implementa el principio de menor privilegio para roles y usuarios.

4. Hardenización del Sistema Operativo

4.1 Aplicación de Parches

Usa AWS Systems Manager para aplicar automáticamente parches y actualizaciones del sistema operativo. Configura parches automáticos para minimizar vulnerabilidades.

1. Automatiza parches con AWS Systems Manager:
    - Crea una "Patch Group" y aplica parches de seguridad regularmente.

4.2 Eliminación de Software Innecesario

Elimina software, controladores y servicios que no sean necesarios. Esto reduce la superficie de ataque y mejora el rendimiento.

1. Usa yum remove o apt remove para eliminar software no necesario.

4.3 Cifrado del Almacenamiento

Asegura que el almacenamiento local esté cifrado para proteger los datos en reposo. Utiliza AWS Key Management Service (KMS) para gestionar las claves de cifrado.

1. Configura volúmenes EBS cifrados:
    - Usa AWS KMS para gestionar claves.
    - Habilita cifrado en volúmenes de inicio y datos.

5. Hardenización de Bases de Datos (PostgreSQL)

5.1 Control de Acceso

Implementa restricciones de acceso para usuarios privilegiados y utiliza contraseñas seguras. Configura roles y permisos adecuados para limitar el acceso a los datos sensibles.

1. Configura roles y permisos:
    - Usa comandos como CREATE ROLE y GRANT para asignar permisos.
2. Limita el acceso:
    - Actualiza pg_hba.conf para especificar rangos IP permitidos.

5.2 Encriptación de Datos

Cifra los datos tanto en almacenamiento como en transferencia. Utiliza SSL/TLS para cifrar las conexiones a la base de datos y AWS KMS para cifrar los datos en reposo.

1. Cifra conexiones:
    - Habilita SSL en la configuración de PostgreSQL (ssl = on).

2. Cifra datos en reposo:

    - Asegúrate de que el volúmen EBS de la base de datos esté cifrado.

5.3 Eliminación de Cuentas No Utilizadas

Borra cuentas de usuario que ya no sean necesarias para minimizar riesgos. Revisa regularmente las cuentas y permisos para asegurar que solo los usuarios autorizados tengan acceso.

1. Lista cuentas:

    - Usa \du en PostgreSQL para revisar usuarios.

2. Elimina usuarios no necesarios:

    - Ejecuta DROP ROLE <usuario>.

6. Recursos adicionales

6.1 CIS Hardened Images

Considera el uso de imágenes endurecidas de CIS disponibles en AWS Marketplace para asegurar que tu sistema operativo esté configurado según las mejores prácticas de seguridad.

6.2 Amazon Inspector

Utiliza Amazon Inspector para evaluar automáticamente las aplicaciones en busca de vulnerabilidades y desviaciones de las mejores prácticas.

6.3 AWS Security Hub

Centraliza y automatiza la gestión de la seguridad en AWS utilizando AWS Security Hub para obtener una visión integral del estado de seguridad de tu infraestructura.

Implementar estas prácticas de hardenización ayudará a proteger tu chatbot contra amenazas y vulnerabilidades, manteniendo la integridad y seguridad de tus datos y sistemas.

Propuesta de Infraestructura Jerarquizable y Escalable para un Chatbot

Para asegurar que la infraestructura de tu chatbot sea jerarquizable, modular y escalable, es fundamental diseñarla con principios de arquitectura moderna y buenas prácticas de desarrollo. A continuación, se presenta una propuesta general:

1. Arquitectura de Microservicios

- Descripción: Divide la aplicación en microservicios independientes, cada uno responsable de una funcionalidad específica del chatbot (por ejemplo, procesamiento de lenguaje natural, gestión de usuarios, integración con bases de datos).

- Ventajas: Facilita la escalabilidad horizontal, permite actualizaciones y despliegues independientes, y mejora la resiliencia del sistema.

2. Contenedores y Orquestación

- Docker: Utiliza Docker para empaquetar cada microservicio en contenedores, asegurando consistencia y portabilidad.

- Kubernetes: Implementa Kubernetes para la orquestación de contenedores, lo que facilita la gestión de despliegues, escalabilidad automática y recuperación ante fallos.

3. Infraestructura en la Nube

- Amazon Web Services (AWS): Utiliza servicios de AWS para una infraestructura flexible y escalable.

- Elastic Kubernetes Service (EKS): Para gestionar tus contenedores de Kubernetes.

- Elastic Load Balancing (ELB): Para distribuir el tráfico entre los microservicios.

- Amazon RDS: Para bases de datos relacionales escalables y gestionadas.

- Amazon S3: Para almacenamiento de objetos, como archivos de configuración y logs.

- AWS Lambda: Para ejecutar funciones serverless que pueden complementar tu chatbot.

4. Red y Seguridad

- Virtual Private Cloud (VPC): Configura una VPC para aislar tu infraestructura y controlar el tráfico de red.

- Security Groups y Network ACLs: Define reglas de seguridad para controlar el acceso a tus recursos.
- AWS WAF (Web Application Firewall): Para proteger tu aplicación contra amenazas comunes.

5. Monitoreo y Logging

- Amazon CloudWatch: Para monitorear el rendimiento y los logs de tu aplicación.
- ELK Stack (Elasticsearch, Logstash, Kibana): Para análisis avanzado de logs y visualización.

6. CI/CD (Integración y Despliegue Continuos)

- Jenkins/GitLab CI: Configura pipelines de CI/CD para automatizar el proceso de construcción, prueba y despliegue de tus microservicios.
- AWS CodePipeline: Para orquestar el flujo de trabajo de CI/CD en AWS.

7. Escalabilidad y Alta Disponibilidad

- Auto Scaling Groups: Configura grupos de autoescalado para ajustar automáticamente la capacidad de tus instancias según la demanda.
- Multi-AZ Deployment: Despliega tus bases de datos y servicios críticos en múltiples zonas de disponibilidad para alta disponibilidad y tolerancia a fallos.

8. Gestión de Configuración y Secretos

- AWS Systems Manager Parameter Store: Para gestionar configuraciones y secretos de manera segura.
- HashiCorp Vault: Para una gestión avanzada de secretos y políticas de acceso.

Esta infraestructura no solo es jerarquizable y modular, sino que también permite una fácil escalabilidad para adaptarse a las necesidades cambiantes del proyecto.

Estrategia de Backup

1. Tipos de Backup:

- **Completo:** Copia todos los datos cada vez que se realiza el backup. Es el más seguro pero también el más costoso en términos de tiempo y almacenamiento.
- **Incremental:** Solo copia los datos que han cambiado desde el último backup. Es más rápido y requiere menos espacio, pero la restauración puede ser más compleja.
- **Diferencial:** Copia todos los datos que han cambiado desde el último backup completo. Es un equilibrio entre los backups completos e incrementales.

- **Frecuencia:** Dependiendo de la criticidad de los datos, los backups pueden ser diarios, semanales o mensuales.
- **Almacenamiento:** Utilizar almacenamiento en la nube (como AWS S3, Azure Blob Storage) o soluciones locales (NAS, SAN).

2. Cálculos de Costes

- **Almacenamiento:**
  - Calcular el coste del almacenamiento necesario para los backups. Por ejemplo, AWS S3 cobra por GB almacenado.

- **Transferencia de Datos:**
  - Considerar los costes de transferencia de datos hacia y desde el almacenamiento de backup.

- **Software de Backup:**
  - Coste de licencias para software de backup (como Veeam, Acronis).

- **Mantenimiento y Soporte:**
  - Costes asociados al mantenimiento y soporte del sistema de backup.

3. Periodos de Conservación

- **Datos Críticos:** Mantener backups durante un periodo más largo (por ejemplo, 1 año o más) para datos críticos.

- **Datos No Críticos:** Mantener backups durante un periodo más corto (por ejemplo, 3-6 meses).
- **Cumplimiento Normativo:** Asegurarse de cumplir con las regulaciones y políticas de retención de datos aplicables a tu industria.

4. Sensibilidad de los Datos Almacenados

- **Clasificación de Datos:** Clasificar los datos según su sensibilidad (por ejemplo, datos personales, datos financieros, datos operativos).

- **Encriptación:** Implementar encriptación tanto en tránsito como en reposo para proteger los datos sensibles.

- **Acceso Controlado:** Asegurar que solo el personal autorizado tenga acceso a los backups, utilizando controles de acceso y autenticación robustos.

Ejemplo de Estrategia de Backup

1. Backup Completo Semanal: Realizar un backup completo cada domingo.

2. Backup Incremental Diario: Realizar backups incrementales de lunes a sábado.

3. Almacenamiento en la Nube: Utilizar AWS S3 para almacenar los backups, con encriptación habilitada.

4. Retención de Datos: Mantener los backups completos durante 1 año y los incrementales durante 3 meses.

5. Costes Estimados:

- Almacenamiento: 100 GB de datos a $0.023 por GB en AWS S3 = $2.30 al mes.

- Transferencia de Datos: 50 GB de transferencia a $0.09 por GB = $4.50 al mes.

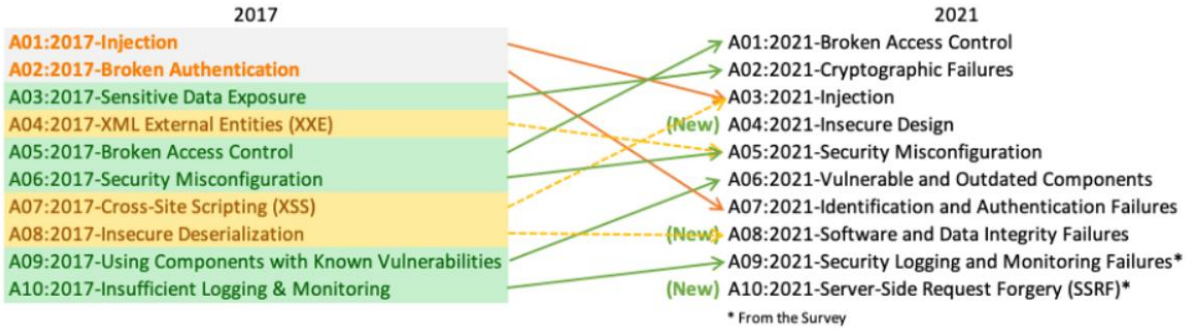
- Software de Backup: Licencia anual de $500.

- Total Mensual Aproximado: $2.30 (almacenamiento) + $4.50 (transferencia) + $41.67 (software) = $48.47 al mes.

OWASP TOP 10

OWASP Top 10 es un documento de concienciación estándar para desarrolladores y seguridad de aplicaciones web. Representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web.

Reconocido mundialmente por los desarrolladores como el primer paso hacia una codificación más segura.



A01:2021: El control de acceso roto sube desde la quinta posición; El 94% de las aplicaciones fueron probadas para detectar algún tipo de control de acceso roto. Las 34 enumeraciones de debilidades comunes (CWE) asignadas al control de acceso roto tuvieron más ocurrencias en las aplicaciones que cualquier otra categoría.

A02:2021: Las fallas criptográficas suben una posición al n.° 2, anteriormente conocida como exposición de datos confidenciales, que era un síntoma general en lugar de una causa raíz. El enfoque renovado aquí está en las fallas relacionadas con la criptografía, que a menudo conducen a la exposición de datos confidenciales o al compromiso del sistema.

A03:2021: La inyección se desliza hacia la tercera posición. El 94% de las aplicaciones fueron probadas para detectar alguna forma de inyección, y los 33 CWE asignados a esta categoría ocupan el segundo lugar con mayor número de apariciones en las aplicaciones. Cross-site Scripting ahora forma parte de esta categoría en esta edición.

A04:2021: Diseño inseguro es una nueva categoría para 2021, que se centra en los riesgos relacionados con fallas de diseño. Si realmente queremos “moverse hacia la izquierda” como industria, es necesario un mayor uso de modelos de amenazas, patrones y principios de diseño seguros y arquitecturas de referencia.

A05:2021: La configuración incorrecta de seguridad asciende desde el puesto 6 en la edición anterior; El 90% de las aplicaciones fueron probadas para detectar algún tipo de configuración incorrecta. Con más cambios hacia software altamente configurable, no sorprende ver que esta categoría suba. La antigua categoría de entidades externas XML (XXE) ahora forma parte de esta categoría.

A06:2021: Componentes vulnerables y obsoletos se titulaba anteriormente. Uso de componentes con vulnerabilidades conocidas y ocupa el puesto número 2 en la encuesta comunitaria de los 10 principales, pero también tenía datos suficientes para llegar al Top 10

mediante el análisis de datos. Esta categoría asciende desde el puesto 9 en 2017 y es un problema conocido que nos cuesta probar y evaluar el riesgo. Es la única categoría que no tiene vulnerabilidades y exposiciones comunes (CVE) asignadas a los CWE incluidos, por lo que en sus puntuaciones se tienen en cuenta un exploit predeterminado y ponderaciones de impacto de 5,0.

A07:2021: Fallos de identificación y autenticación anteriormente era Autenticación rota y se está deslizando hacia abajo desde la segunda posición, y ahora incluye CWE que están más relacionados con fallos de identificación. Esta categoría sigue siendo una parte integral del Top 10, pero la mayor disponibilidad de marcos estandarizados parece estar ayudando.

A08:2021: Fallos de integridad de datos y software es una nueva categoría para 2021, que se centra en hacer suposiciones relacionadas con actualizaciones de software, datos críticos y canalizaciones de CI/CD sin verificar la integridad. Uno de los impactos ponderados más altos de los datos de Vulnerabilidad y Exposiciones Comunes/Sistema de Puntuación de Vulnerabilidad Común (CVE/CVSS) asignados a los 10 CWE en esta categoría. La deserialización insegura de 2017 ahora forma parte de esta categoría más amplia.

A09:2021: Fallas de registro y monitoreo de seguridad anteriormente era Registro y monitoreo insuficientes y se agrega de la encuesta de la industria (n.° 3), subiendo desde el n.° 10 anterior. Esta categoría se amplía para incluir más tipos de fallas, es difícil de probar y no está bien representada en los datos CVE/CVSS. Sin embargo, las fallas en esta categoría pueden afectar directamente la visibilidad, las alertas de incidentes y el análisis forense.

A10:2021: Se agrega la falsificación de solicitudes del lado del servidor de la encuesta de la comunidad Top 10 (n.º 1). Los datos muestran una tasa de incidencia relativamente baja con una cobertura de pruebas superior al promedio, junto con calificaciones superiores al promedio de potencial de explotación e impacto. Esta categoría representa el escenario en el que los miembros de la comunidad de seguridad nos dicen que esto es importante, aunque no esté ilustrado en los datos en este momento.

Checklist

En términos generales, es mucho menos costoso construir software seguro que corregir problemas de seguridad cuando ha sido completado, sin mencionar los costos que pueden estar asociados a un quiebre en la seguridad.

Asegurar recursos críticos de software se ha vuelto más importante que nunca debido a que el foco de los atacantes se ha desplazado hacia la capa de aplicación. Un estudio en 2009 de la SANS1 determinó que los ataques contra aplicaciones web constituyen más del 60% del total de intentos de ataque observados en la Internet.

Para asegurar el cumplimiento de los estándares de seguridad, hemos implementado una lista de verificación basada en las mejores prácticas de codificación segura. Esta lista abarca varios aspectos críticos de la seguridad en el desarrollo de software y está diseñada para mitigar vulnerabilidades comunes.

Validación de entradas

1. Realizar validaciones en sistemas confiables: Realizar todas las validaciones de datos en el servidor.

2. Identificación de fuentes de datos: Clasificar las fuentes como confiables o no confiables y validar todas las no confiables.

3. Rutina de validación centralizada: Centralizar la validación de datos de entrada.

4. Rechazo de fallas de validación: Las fallas deben resultar en el rechazo de los datos de entrada.

5. Validación de parámetros de entrada: Incluir parámetros, URLs y cabeceras HTTP, y validar datos redireccionados y tipos de datos no esperados.

6. Verificación de caracteres peligrosos: Validar caracteres peligrosos como <, >,&, etc., y comprobar la existencia de bytes nulos y caracteres de nueva línea.

Codificación de salidas

1. Codifique todos los caracteres salvo que sean reconocidos como seguros por el interpretador al que están destinados.

2. Contextualice la codificación de salida de todos los datos devueltos por el cliente que se originen desde fuera de la frontera de confianza de la aplicación. La codificación de entidades HTML es un ejemplo, aunque no sea suficiente en todos los casos.

3. Sanitizar contextualmente todos las salidas de datos no confiables hacia consultas SQL, XML y LDAP.

Administración de autenticación y contraseñas

1. Requerir autenticación para todos los recursos y páginas excepto aquellas específicamente clasificadas como públicas.

2. Todos los controles de autenticación deben fallar de una forma segura.

3. Utilizar únicamente pedidos del tipo HTTP POST para la transmisión de credenciales de autenticación.

4. Hacer cumplir por medio de una política o regulación los requerimientos de complejidad de la contraseña. Las credenciales de autenticación deben ser suficientes como para resistir aquellos ataques típicos de las amenazas en el entorno del sistema. Ej: obligar el uso de combinaciones de caracteres

numéricos/alfanuméricos y/o caracteres especiales.

5. Hacer cumplir por medio de una política o regulación los requerimientos de longitud de la contraseña. Comúnmente se utilizan ocho caracteres, pero dieciséis es mejor, adicionalmente considerar el uso de frases de varias palabras.

6. No se debe desplegar en la pantalla la contraseña ingresada. A modo de ejemplo, en formularios web, utilizar el tipo de entrada “password” (input type “password”).

7. El cambio y reseteo de contraseñas requieren los mismos niveles de control como aquellos asociados a la creación y autenticación de cuentas.

8. Deshabilitar la funcionalidad de “recordar” campos de contraseñas.

Administración de sesiones

1. Utilizar los controles del servidor o del framework para la administración de sesiones. La aplicación solo debe reconocer estos identificadores como válidos.

2. Los controles de administración de sesiones deben utilizar algoritmos que generen identificadores suficientemente aleatorios

3. Establecer un tiempo de vida de la sesión lo más corto posible, balanceando los riesgos con los requerimientos del negocio. En la mayoría de los casos, nunca debería ser superior a varias horas.

4. No exponer identificadores de sesión en URLs, mensajes de error ni logs. Los identificadores de sesión solo deben ser ubicados en la cabecera de la cookie HTTP. A modo de ejemplo, no transmitir el identificador de sesión como un parámetro GET.

5. Manejo de sesión complementario para operaciones sensibles del lado del servidor, como pueden ser: gestión de cuentas o utilizando tokens o parámetros por sesión. Este método puede ser utilizado para prevenir ataques de Cross Site Request Forgery Attacks (Falsificación de petición en sitios cruzados, conocido por sus siglas CSRF).

6. Configurar las cookies con el atributo HttpOnly, salvo que se requiera específicamente scripts del lado del cliente en la aplicación, para leer o configurar una cookie.

Control de Acceso

1. Utilizar únicamente objetos confiables del sistema. Por ejemplo: objetos de sesión del servidor para la toma de decisiones de autorización.

2. Denegar todos los accesos en caso de que la aplicación no pueda acceder a la información de configuración de seguridad.

3. Restringir el acceso a URLs protegidas, solo a usuarios autorizados.

4. Restringir el acceso a funciones protegidas, solo a usuarios autorizados.

5. Restringir las referencias directas a objetos, solo a usuarios autorizados.

6. Restringir el acceso a servicios, solo a usuarios autorizados.

7. Restringir el acceso a información de la aplicación, solo a usuarios autorizados.

8. Restringir el acceso a usuario, atributos y política de información utilizada por los controles de acceso.

Prácticas Criptográficas

1. Proteger secretos maestros (master secrets) del acceso no autorizado.

2. Los módulos de criptografía deberían en caso de falla, fallar en forma segura.

3. Establecer y utilizar una política y un proceso de cómo manejar las claves criptográficas.

Manejo de Errores y Logs

1. No difundir información sensible en respuestas de error, incluyendo detalles del sistema, identificadores de sesión o información de la cuenta.

2. Utilizar manejadores de errores que no muestren información de debugging o de memoria.

3. Implementar mensajes de error genéricos y utilizar páginas de error adaptadas.

4. Liberar espacio de memoria en cuanto una condición de error ocurra.

5. Registrar en un log todos los eventos de intento de evasión de controles, incluyendo cambios en el estado de la información no esperados.

6. Registrar en un log todos los intentos de conexión con tokens inválidos o vencidos.

7. Registrar en un log todas las funciones administrativas, incluyendo cambios en la configuración de seguridad.

8. Utilizar una función de hash para validar la integridad de los logs.

Protección de Datos

1. Implementar el mínimo privilegio, restringir el acceso de los usuarios solamente a la funcionalidad, datos y sistemas de información que son necesarios para realizar sus tareas.
2. Remover cualquier aplicación que no sea necesaria y la documentación de los sistemas que pueda revelar información útil para los atacantes.

Seguridad en las Comunicaciones

1. Implementar encriptación para todas las transmisiones de información sensible. Esto debería incluir TLS para proteger la conexión y se puede combinar con una encriptación discreta de archivos sensibles en conexiones que no estén basadas en HTTP.

2. Los certificados TLS deben de ser válidos y contener el nombre de dominio correcto, no deben estar expirados y deberán ser instalados con los certificados intermedios si son requeridos.

3. Filtrar los parámetros que contengan información sensible de los referer HTTP, cuando existen vínculos a sitios externos.

4. Especificar los caracteres de codificación para todas las conexiones.

Configuración de los Sistemas

1. Asegurarse de que los servidores, los frameworks y los componentes del sistema están corriendo la última versión aprobada.

2. Restringir el servidor web, los procesos y las cuentas de servicios con el mínimo privilegio posible.

3. Remover código de testeo o cualquier funcionalidad que no sea tenida en cuenta en producción, previo a realizar la puesta en producción.

4. Definir cuáles de los métodos HTTP, GET o POST, la aplicación va a soportar y si deben de ser manejados de forma diferente en las distintas páginas de la aplicación.

Seguridad de Base de Datos

1. La aplicación debe de utilizar el mínimo nivel de privilegios cuando accede a la base de datos.

2. Utilice credenciales seguras para acceder a la base de datos.

3. Cierre la conexión a la base de datos tan pronto como sea posible.

4. Remueva o cambie todas las contraseñas administrativas por defecto. Utilice

contraseñas fuertes o frases o implemente autenticación de múltiples factores.

5. Deshabilitar las cuentas por omisión que no son necesarias para la operativa del negocio.

Manejo de Archivos

1. No utilizar directamente información provista por el usuario en ninguna operación dinámica.

2. Exigir autenticación antes de permitir la transferencia de un archivo al servidor.

3. Validar los tipos de archivo transferidos verificando la estructura de los encabezados. La validación del tipo de archivo únicamente por la extensión no es suficiente.

4. No incluir en parámetros nombres de directorios o rutas de archivos, en su lugar utilizar índices que internamente se asocien a directorios o rutas predefinidas.

Manejo de Memoria

1. Utilice controles en la entrada y la salida de información no confiable.

Prácticas Generales para la Codificación

1. Utilice las APIs previstas para el acceso a funciones específicas del sistema operativo. No permitir que la aplicación ejecute comandos directamente en el sistema operativo, menos aún mediante la invocación de una shell.

2. Proteja las variables y recursos compartidos de accesos concurrentes inadecuados.

3. Revisar todas las aplicaciones secundarias, código provisto por terceros y bibliotecas para determinar la necesidad del negocio para su utilización y validar el funcionamiento seguro, ya que estos pueden introducir nuevas vulnerabilidades.

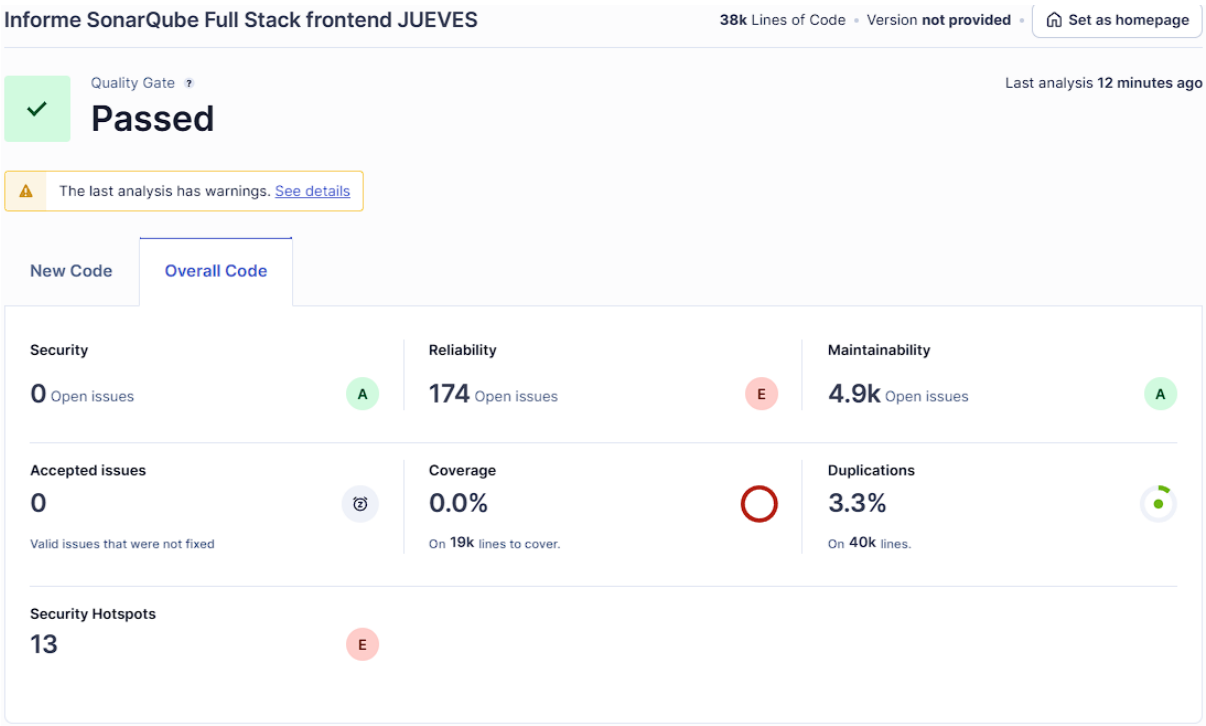
Para asegurar una estructura clara y cohesiva que resalte la importancia de la seguridad web, hemos compartido estas directrices con los equipos de Full Stack y Data Science. Esto garantiza que todos los miembros del equipo comprendan e implementen estas prácticas de seguridad. Al final del informe, incluiremos una lista de verificación detallada para facilitar la verificación y el seguimiento de estas acciones.

Pruebas de código estático con SonarQube

SonarQube es una herramienta de análisis de código estático que ayuda a los equipos de desarrollo a mantener y mejorar la calidad del código. Evalúa el código fuente en busca de errores, vulnerabilidades de seguridad y problemas de estilo, proporcionando informes detallados y recomendaciones para resolverlos.

Con SonarQube, identificamos defectos y riesgos potenciales en la aplicación durante el proceso de desarrollo. Esto nos permitió abordar vulnerabilidades de seguridad antes de que llegaran al despliegue de la aplicación, mejorando la protección general y la integridad de la misma. La herramienta se integró fácilmente en nuestro flujo de trabajo, asegurando que el código estuviera limpio y protegido contra posibles amenazas.

A lo largo del desarrollo del Desafío de Tripulaciones, realizamos múltiples análisis de código con SonarQube. Estos análisis nos proporcionaron una visión clara de los problemas presentes en el código y nos guiaron en la implementación de soluciones efectivas para mejorar la calidad y seguridad de nuestra aplicación.



Conclusiones

A lo largo de este trabajo, se ha explorado la importancia de adoptar prácticas y herramientas clave para garantizar un entorno de desarrollo y operación robusto, seguro y escalable en proyectos de chatbots. En este sentido, se destacó la relevancia de coordinar equipos multidisciplinarios para abordar de manera conjunta los desafíos de seguridad, rendimiento y fiabilidad, elementos fundamentales en cualquier proyecto tecnológico complejo. La integración efectiva de estos equipos facilita la identificación de riesgos potenciales y la implementación de soluciones colaborativas, lo que permite una mejora continua en el ciclo de desarrollo y la mitigación de vulnerabilidades.

Una de las estrategias fundamentales para mejorar la seguridad de la infraestructura de un chatbot es el modelado de amenazas, herramienta que permite a los equipos de desarrollo mapear posibles vectores de ataque y priorizar los esfuerzos de mitigación. A través de este proceso, se logra anticipar amenazas, lo cual es crucial para la prevención de fallos que puedan comprometer la integridad del sistema. Complementariamente, el hardening o endurecimiento de sistemas refuerza esta seguridad, aplicando configuraciones y políticas que reducen la superficie de ataque, desde la gestión de parches hasta la configuración de firewalls y el uso de autenticación robusta. Ambas prácticas, el modelado de amenazas y el hardening, no solo mejoran la seguridad, sino que también contribuyen a una infraestructura más resistente y menos susceptible a compromisos.

Otro aspecto clave es la infraestructura de la que depende el chatbot, la cual debe ser escalable y flexible para soportar picos de demanda sin afectar el rendimiento. Utilizar arquitecturas jerarquizables y escalables, como las basadas en microservicios y contenedores, permite manejar grandes volúmenes de usuarios sin sacrificar la eficiencia. Esta capacidad de escalar de manera horizontal y vertical asegura que el chatbot pueda responder adecuadamente a cambios en la demanda sin poner en riesgo la experiencia del usuario o la estabilidad del servicio. Esta infraestructura debe, además, contar con una estrategia de backup sólida que garantice la continuidad operativa frente a posibles fallos, ataques o pérdidas de datos. Los backups deben ser regulares, cifrados y fácilmente restaurables, protegiendo tanto la información sensible de los usuarios como los elementos críticos del sistema.

Integrar prácticas de seguridad reconocidas, como las del OWASP Top 10, es esencial para proteger el sistema contra amenazas comunes en aplicaciones web, como la inyección de SQL, la exposición de datos sensibles o la autenticación insegura. Asegurar que el chatbot esté alineado con estas mejores prácticas permite mitigar riesgos y proteger tanto la integridad de los datos como la confidencialidad de las interacciones con los usuarios. Esta protección se complementa con el uso de herramientas de análisis de código estático, como SonarQube, que identifican vulnerabilidades y errores de programación antes de que el código llegue a producción. Las pruebas de código estático mejoran la calidad del software, asegurando que se sigan los estándares de seguridad y optimizando el rendimiento del sistema en su conjunto.

En conjunto, la implementación de estas prácticas y herramientas ofrece una base sólida para el desarrollo y operación de chatbots. Desde la planificación de la seguridad hasta la gestión de la infraestructura y la calidad del código, todas estas estrategias se entrelazan para crear un entorno más seguro, eficiente y resiliente. La integración de enfoques de seguridad proactiva, como el modelado de amenazas, el hardening, la estrategia de backup, las prácticas de OWASP y las pruebas de código estático, minimiza riesgos y asegura que el sistema no solo sea escalable y funcional, sino también confiable y seguro para los usuarios.