.UBAfiuba
FACULTAD DE INGENIERÍA

INFORME REINGENIERÍA
DEL SISTEMA Z

Grupo 8

Integrantes:

Nombre	Padrón	Mail
Clara Ruano Frugoli	106835	cruano@fi.uba.ar
Gabriel Katta	105935	gkatta@fi.uba.ar
Maria Paula Brück	107533	pbruck.ext@fi.uba.ar
Paolo Belforte	109432	pbelforte@fi.uba.ar
Rubén Bohórquez	109442	rbohorquez@fi.uba.ar
Ramiro Gestoso	105950	rgestoso@fi.uba.ar
Juan Ignacio Medone Sabatini	103878	jmedone@fi.uba.ar

Índice

Introducción

Luego de completar las tareas de investigación y el pentest correspondiente sobre el sistema Z de la empresa XY, se generaron los siguientes entregables:

- Pentest - Informe Técnico
- Pentest - Informe Ejecutivo

Una vez entregados los informes, el equipo elaboró este documento con el objetivo de especificar las modificaciones, ajustes y correcciones necesarias para fortalecer la seguridad del sistema. Además, se incluye una explicación detallada de cada solución propuesta y su relación con los principios de la seguridad informática.

Puntos a analizar para la Reingeniería

Principio	Problema	Solución
Disponibilidad	- El sistema actual puede sufrir caídas o lentitud extrema cuando hay múltiples solicitudes. - Servidor con OS sin soporte (CentOS 5).	- Rate Limiting (límite de peticiones). - Web Application Firewall (WAF). - Balanceador de Carga. - Migrar a SO soportado (Linux) con parches de seguridad.
Autorización	- Cualquier usuario puede acceder a expedientes ajenos. - No hay tokens anti CSRF.	- RBAC (*Role-Based Access Control)* + verificación de propiedad en cada solicitud. - Implementar tokens CSRF en formularios.
Integridad	- Creación de expedientes corruptos (sin código o datos inválidos). - Reutilización de tokens de sesión antiguos.	- Validar entradas en backend. - Implementar JWT con expiración corta y renovación obligatoria.
Autenticación	- No hay protección contra fuerza bruta (sin bloqueo por IP, CAPTCHA o delays). - Credenciales de prueba activas ("pedro - pedro"). - Sesión no se invalida al acceder a /login.	- Implementar rate limiting y bloqueo temporal tras intentos fallidos. - Eliminar credenciales predeterminadas. - Invalidar sesión al navegar a /login y redirigir a home.
Confidencialidad	- Cookies de sesión mal configuradas (httpOnly=false, secure=false,	- Configurar cookies: httpOnly=True, secure=True, SameSite=Strict. - Implementar RBAC

	SameSite=None). - Acceso a expedientes de otros usuarios. - Comunicación HTTP sin cifrado (no HTTPS).	(control de acceso por roles). - Migrar a HTTPS/TLS 1.3.
Criptografía	- Algoritmos SSH obsoletos (DSA, SHA-1, CBC). - Vulnerabilidad Ghostcat (CVE-2020-1938) en Tomcat.	- Actualizar SSH: deshabilitar DSA, usar Ed25519 o ECDSA. - Parchear Tomcat o deshabilitar AJP.
Trazabilidad	- No hay logs claros para timeouts o acciones críticas. - Exposición de cookies en errores HTTP 400.	- Implementar SIEM (*Security Information and Event Management)* para monitoreo. - Configurar páginas de error personalizadas.
Protección de Servicios	- Puertos innecesarios abiertos (3306/MySQL, 8009/AJP). - Métodos HTTP TRACE/TRACK habilitados.	- Hardening: Cerrar puertos no esenciales (RPCBind, AJP). - Deshabilitar métodos HTTP inseguros en el servidor web.

Detalles técnicos de Reingeniería

Este apartado detalla cómo se implementarán las soluciones técnicas para corregir las vulnerabilidades reportadas.
Incluye configuraciones específicas, herramientas y pasos prácticos para cada problema de seguridad identificado.
Su objetivo es guiar la ejecución concreta de las mejoras necesarias.

Análisis de Mejoras

Infraestructura

Migración del sistema operativo

Problema:
CentOS 5 está descontinuado y ya no recibe actualizaciones de seguridad ni soporte.

Solución:

- **Opción recomendada: Red Hat Enterprise Linux (RHEL) 10**: soporte hasta 2035.
- **Pasos:**
    a. Realizar un **backup completo** de datos y configuraciones actuales.
    b. Instalar el nuevo sistema operativo en un entorno de **pruebas controlado**.
    c. Migrar servicios de forma **escalonada**, validando su compatibilidad (uso de máquinas virtuales o contenedores Docker).
    Actualizar **dependencias, librerías y parches críticos**.
    d. Documentar el proceso para facilitar mantenimientos futuros.

Separación de Ambientes

Problema:
Los ambientes de desarrollo, testeo y producción no están correctamente aislados, lo que puede generar interferencias y riesgos de seguridad.

Solución:

- Usar **máquinas virtuales** o **contenedores Docker/Kubernetes** para crear entornos independientes.
- Establecer reglas estrictas de **acceso y despliegue** para cada ambiente.
- Implementar **versionado de código y CI/CD**, asegurando que solo código validado pase a producción.
- Limitar la conexión entre entornos; por ejemplo, evitar que el testeo pueda acceder a bases de datos de producción.

Migración hacia Arquitectura Zero Trust

Problema:
Actualmente, la red y los servicios operan bajo un modelo de confianza implícita, lo cual deja vulnerabilidades abiertas frente a accesos no autorizados internos o comprometidos.

Solución:

- **Objetivo:**
  Adoptar un modelo de **Zero Trust**, donde "nada se confía por defecto", ni dentro ni fuera del perímetro de red.
- **Pasos clave:**
  - Implementar **autenticación multifactor (MFA)** para todos los accesos.
  - Aplicar **segmentación de red** y control de acceso basado en identidad.
  - Monitorear y registrar **todas las actividades** en la red.
  - Verificar continuamente el estado de dispositivos y usuarios antes de otorgar acceso.
  - Usar herramientas como **reverse proxies**, **gateways de acceso seguro** o plataformas como **Tailscale.**

Web y Backend

Eliminación de AJP y Vulnerabilidad Ghostcat

Problema:
El uso del conector AJP en Tomcat puede exponer el sistema a la vulnerabilidad **Ghostcat (CVE-2020-1938)**, que permite la lectura arbitraria de archivos del servidor.

Solución:

- **Eliminar AJP** como método de comunicación entre Apache y Tomcat.
- Usar **HTTP o HTTPS estándar** como canal interno.
- Configurar **mod_proxy_http** en Apache o, preferentemente, utilizar **Nginx como reverse proxy**, por su **simplicidad, rendimiento y mejor manejo de carga**.
- Configurar Tomcat para escuchar únicamente en **localhost** o en un **puerto interno** no accesible desde el exterior.

Migración a HTTPS

Problema:
La transmisión de datos sin cifrar expone el tráfico a ataques de tipo sniffing, MITM y manipulación de paquetes.

Solución:

- Instalar certificados **SSL/TLS válidos y corporativos**.
- Redirigir automáticamente todo tráfico HTTP a **HTTPS**.
- Configurar el servidor web con **TLS 1.2 o superior** y deshabilitar versiones obsoletas.
- Hacer pruebas de configuración con herramientas como **SSL Labs** para asegurar un cifrado robusto.

Deshabilitar Métodos HTTP Inseguros

Problema:
Métodos como **TRACE** o **TRACK** pueden permitir ataques como **Cross-Site Tracing (XST)**.

Solución:

- Deshabilitar explícitamente métodos HTTP inseguros en la configuración del servidor web.

Separación de Capas (Backend y Presentación)

Problema:
Una arquitectura monolítica expone más superficie de ataque y dificulta la implementación de controles de acceso segmentados.

Solución:

- Separar la lógica de negocio en un backend independiente, accedido sólo mediante una **API RESTful**.
- Implementar mecanismos de autenticación como **tokenización segura (JWT, OAuth2)**.
- Evitar exposición directa del backend: todo acceso pasa por el **frontend (UI o API Gateway)**.

Base de Datos

Base de Datos Dedicada

Problema:
Compartir una base de datos entre varios sistemas aumenta la superficie de riesgo y puede generar accesos cruzados no autorizados.

Solución:

- Crear una **instancia de base de datos exclusiva** para el sistema Z.
- Asegurar que esté **aislada lógicamente o en una red privada**, sin acceso público.

Control de Acceso Basado en Roles

Problema:
El uso de cuentas con privilegios excesivos puede derivar en escalamiento de privilegios o destrucción de datos accidentales.

Solución:

- Aplicar el **principio de menor privilegio**: cada cuenta solo debe tener acceso a las funciones necesarias.
- Usar **roles bien definidos** para cada tipo de usuario o servicio.
- Deshabilitar accesos innecesarios (por ejemplo, acceso remoto root desactivado).

Cifrado de Datos Sensibles

Problema:
Datos sensibles pueden ser comprometidos si no están debidamente cifrados, tanto en reposo como en tránsito.

Solución:

- **Cifrado en tránsito:** usar **TLS** en todas las conexiones a la base de datos.
- **Cifrado en reposo:** implementar mecanismos nativos del motor de base de datos o usar librerías de cifrado a nivel de aplicación para campos específicos (cómo contraseñas, tokens, información personal sensible).

Auditoría y Trazabilidad

Problema:
La falta de auditoría deja al sistema sin capacidad de identificar accesos indebidos o fallas de seguridad.

Solución:

- Activar el logging **detallado** de accesos a la base de datos.
- Mantener los logs en un sistema centralizado y seguro.
- Usar **firmas digitales o hashes** para garantizar la integridad de los logs.
- Configurar alertas automáticas para accesos sospechosos o cambios en datos críticos.

Autenticación y Autorización

Protección contra Ataques de Fuerza Bruta

Problema:
El sistema es vulnerable a intentos automatizados de login que pueden derivar en acceso no autorizado.

Solución:

- Implementar **rate-limiting** por IP y/o usuario.
- Usar **bloqueo progresivo** tras múltiples intentos fallidos.
- Añadir **CAPTCHA** tras cierto número de fallos.
- Registrar e **informar intentos sospechosos** al equipo de seguridad.

Encriptado Seguro de Contraseñas

Problema:
El almacenamiento inseguro de contraseñas puede llevar a su filtración en caso de compromiso.

Solución:

- Usar algoritmos de hash modernos y seguros como:
    - **bcrypt**
    - **Argon2** (preferido por su resistencia a ataques de canal lateral y su flexibilidad).

Gestión Segura de Sesiones

Problema:
Las sesiones persistentes o mal gestionadas pueden permitir secuestro de sesión o uso de credenciales inválidas.

Solución:

- Invalidar automáticamente la sesión al:
  - Cerrar sesión.
  - Reingresar al endpoint `/login`.
- Usar **cookies seguras**:
  - `Secure`: solo se envía por HTTPS.
  - `HttpOnly`: inaccesible desde JavaScript.
  - `SameSite=Strict`: evitar envío entre sitios.

Gestión de Tokens de Autenticación

Problema:
Los tokens que no caducan ni se invalidan correctamente pueden ser reutilizados en ataques.

Solución:

- Usar **JWTs con expiración corta** y mecanismos de renovación (rotating tokens o refresh tokens).
- Implementar una **blacklist o tabla de revocación** para tokens invalidados antes de su expiración.
- **Evitar reuso de tokens** al cerrar sesión: el token debe ser descartado activamente.

Soporte para Autenticación de Dos Factores (2FA)

Problema:
El acceso basado únicamente en usuario y contraseña es fácilmente comprometible.

Solución:

- Implementar **2FA obligatorio** para cuentas sensibles.
- Métodos recomendados:
  - **TOTP (Time-based One-Time Passwords)** con apps como Google Authenticator o Authy.
  - **Envío de códigos por correo o SMS** (menos seguro, pero útil como paso inicial).
- Asegurar la recuperación segura de cuentas si se pierde el segundo factor.

Control de Acceso

Modelo de Roles (RBAC)

Problema:
La falta de segmentación de permisos permite que usuarios accedan a funcionalidades que no les corresponden.

Solución:

- Implementar **RBAC (Role-Based Access Control)** con permisos claros por perfil:
  - Ej.: admin, analista, gestor, auditor, etc.

- Asociar cada endpoint y acción a un rol permitido.
- Mantener una matriz de permisos actualizada.

Validación de Sesión Activa

Problema:
Los endpoints podrían estar expuestos a accesos sin autenticación válida.

Solución:

- Validar que **cada petición a endpoints sensibles** (crear, editar, consultar expedientes, etc.):
    - Esté autenticada.
    - Corresponde a una **sesión activa** y válida.
    - Tenga los permisos adecuados según su rol.

Auditoría y Registro de Accesos

Problema:
Sin trazabilidad, los intentos de acceso indebido pueden pasar desapercibidos.

Solución:

- Loggear todos los accesos a recursos sensibles.
- Registrar **intentos fallidos o denegados** con detalles (usuario, IP, timestamp).
- Implementar alertas ante **patrones anómalos** o intentos reiterados.

Monitoreo y Hardening

Eliminación de Servicios y Puertos Innecesarios

Problema:
Cada servicio expuesto innecesariamente representa una potencial vía de ataque.

Solución:

- Identificar servicios activos con herramientas como `netstat`, `ss`, `nmap`.
- Deshabilitar o eliminar servicios no utilizados.
- Cerrar puertos no requeridos desde el firewall.
- Configurar el sistema operativo para que solo escuche en interfaces necesarias.

Fortalecimiento del Acceso por SSH

Problema:
Configuraciones inseguras en SSH pueden facilitar accesos indebidos o ataques criptográficos.

Solución:

- Deshabilitar algoritmos obsoletos y vulnerables:
    - **Desactivar DSA, RC4, CBC.**
- Usar únicamente algoritmos modernos:
    - **ed25519** para claves.
    - **chacha20-poly1305** para cifrado.

Monitoreo con IDS/IPS

Problema:
La falta de detección temprana de ataques impide actuar a tiempo ante incidentes.

Solución:

- Instalar y configurar un **IDS/IPS como Snort o Suricata**.
- Integrar logs.
- Definir alertas para patrones sospechosos (escaneo de puertos, cambios en archivos, acceso fuera de horario).

Escaneos y Pentesting Periódico

Problema:
Sin validaciones continuas, las nuevas vulnerabilidades podrían pasar desapercibidas.

Solución:

- Automatizar escaneos de vulnerabilidades:
  - **OpenVAS**, **Nessus** en fase de test o staging.
- Realizar **pentesting manual periódico**, sobre todo tras actualizaciones o cambios arquitectónicos.
- Corregir vulnerabilidades detectadas según criticidad (CVSS o metodología OWASP).

Despliegue y DevOps

CI/CD Seguro con Pruebas de Seguridad

Problema:
El código malicioso o inseguro podría desplegarse sin detección previa.

Solución:

- Integrar herramientas de seguridad en el pipeline:
  - **SAST** (ej.: SonarQube, Semgrep) → código fuente.
  - **DAST** (ej.: OWASP ZAP, Burp Suite) → ejecución.
  - **SCA** (ej.: Snyk, OWASP Dependency-Check) → librerías.
- Automatizar ejecuciones en cada build o pull request.

Validación Rigurosa de Inputs y Outputs

Problema:
La falta de validaciones puede permitir ataques como XSS, SQLi, SSRF, etc.

Solución:

- Validar **todo input del usuario** (formato, longitud, tipo, valores esperados).
- Sanitizar y escapar outputs.
- Usar **librerías de validación seguras**, evitar expresiones regulares inseguras o parseo manual.

Producción Libre de Credenciales de Prueba

Problema:
Usuarios de test o credenciales hardcodeadas pueden ser utilizadas maliciosamente.

Solución:

- Escanear el código y los entornos en busca de:
  - Usuarios `admin`, `test`, `dev`.
  - Tokens o claves visibles (con herramientas como `truffleHog`, `gitleaks`).
- Eliminar todo lo que no sea real y necesario antes del despliegue.

Gestión Segura de Secretos

Problema:
El manejo inadecuado de claves o contraseñas puede generar filtraciones.

Solución:

- Usar gestores de contraseñas/secretos como **1Password**, **HashiCorp Vault**, **AWS Secrets Manager**.
- Las credenciales a servicios críticos deben ser periódicamente rotadas para mitigar filtraciones accidentales.
- Prohibir el uso de documentos, hojas de cálculo o archivos `.env` sin cifrado como medio de almacenamiento.
- Compartir secretos por canales seguros, con expiración y acceso limitado.

Arquitectura de Red

Segmentación de Red

Problema:

Tener todo en una única red facilita la propagación de un ataque interno.

Solución:

- Separar entornos en VLANs:
  - Usuarios
  - Servidores (aplicación, base de datos)
  - Administración
- Aplicar políticas estrictas de comunicación entre VLANs vía firewall.

Zona Desmilitarizada (DMZ)

Problema:
Tener servidores expuestos en la misma red que recursos internos es altamente riesgoso.

Solución:

- Ubicar el **servidor web en una DMZ**, intermedia entre internet y la red interna.
- Asegurar que solo pueda comunicarse con el backend o base de datos en puertos específicos y limitados.
- Loggear y monitorear todo tráfico entre zonas.

Seguridad en la Red WiFi

Problema:
Una red WiFi abierta o mal autenticada puede ser explotada por intrusos cercanos.

Solución:

- Implementar **WPA2 Enterprise con servidor RADIUS**.
- Usar certificados o credenciales individuales por empleado.

- Monitorear accesos por MAC, logs y autenticaciones fallidas.

Control y Monitoreo de Tráfico Interno

Problema:
El movimiento lateral dentro de la red puede pasar desapercibido sin monitoreo interno.

Solución:

- Registrar tráfico entre segmentos críticos, especialmente hacia la base de datos.
- Usar herramientas de inspección como **Wireshark** en entornos controlados.
- Aplicar reglas en el firewall para limitar el tráfico a lo estrictamente necesario.

Deshabilitar Respuestas ICMP Timestamp

Problema:
El sistema responde a paquetes ICMP tipo 13 (Timestamp Request), lo cual permite a un atacante externo conocer la hora del sistema y realizar fingerprinting, aumentando la superficie de ataque.

Solución:

- Bloquear solicitudes ICMP tipo 13 en el firewall
- Validar también que el sistema no responda a otras solicitudes ICMP innecesarias, como address-mask-request

Conclusión

Las implementaciones técnicas detalladas en este informe permitirán corregir el 100% de las vulnerabilidades críticas identificadas en el Sistema Z, fortaleciendo su postura de seguridad mediante:

1. Control de Accesos Estricto
    - Implementación de RBAC (Role-Based Access Control) para garantizar que los usuarios solo accedan a los recursos necesarios.
    - Autenticación Multifactor (MFA) para prevenir accesos no autorizados, incluso en caso de robo de credenciales.
2. Protección Integral de Datos
    - Cifrado de datos en reposo (AES-256) y en tránsito (HTTPS/TLS 1.3) para evitar fugas de información.
    - Eliminación de protocolos inseguros (AJP, SSH obsoleto) y configuración de certificados SSL/TLS válidos.
3. Detección y Respuesta Proactiva
    - Monitoreo continuo con SIEM para identificar amenazas en tiempo real.
    - IDS/IPS (Suricata, Snort) para bloquear intentos de intrusión.
    - Auditorías periódicas con herramientas como OpenVAS y pentesting manual.

Recomendaciones a Futuro

Para mantener la seguridad del Sistema Z en el tiempo, se recomienda:

- **Capacitación continua** del personal en desarrollo seguro y gestión de incidentes.

- **Aplicar estándares reconocidos** para guiar mejoras futuras.
- **Automatizar controles de seguridad** en CI/CD, incluyendo análisis de código y validación de configuraciones.
- **Gestionar vulnerabilidades de forma proactiva**, con escaneos periódicos y aplicación de parches bajo control.
- **Realizar revisiones post-incidente**, documentando aprendizajes y reforzando controles.

Estas acciones complementan las medidas implementadas y permiten sostener una postura de seguridad robusta frente a amenazas cambiantes.