Security Analysis of SSL/TLS Configuration Gap in Django-Based Industrial Embedded Systems: A Code-Level Assessment

Benjamín Chuquimango[1,2], and Francisco Hernán Ledesma [1]

[1] Instituto Nacional de Tecnología Industrial (INTI), San Martín, Buenos Aires, Argentina
[2] Maestría en Seguridad Informática, Universidad de Buenos Aires, Buenos Aires, Argentina
benjaminchuq@gmail.com, fhledesma@gmail.com

Abstract. Industrial vision systems built on embedded hardware and rapid-development web frameworks present a particular attack surface that has received limited attention in the regional literature. This paper reports a security analysis of PLABVision, an in-line product classification system developed through a collaboration between an applied research institute and an Argentine manufacturing SME, running on an NVIDIA Jetson Orin Nx platform with an industrial GigE camera and a Django 4.2 backend. The evaluation was conducted on the production codebase in an equivalent development environment, combining static configuration analysis with dynamic testing using OWASP ZAP. The main finding is an SSL/TLS configuration gap between the development environment and the production deployment: the system correctly configures HTTPS at the Django application layer, but the nginx server handling real traffic does not have TLS enabled, resulting in credentials being transmitted in plaintext. This was demonstrated by intercepting an authentication request with OWASP ZAP, capturing unencrypted credentials over HTTP. The evaluation was carried out in a local development environment; validation of the attack vector on the actual industrial LAN remains future work. The results are applicable to other SME projects adopting similar Django-nginx-embedded hardware stacks, a pattern that is increasingly common in Argentine industry.

Keywords: SSL/TLS misconfiguration, SME cybersecurity, Django web security, industrial embedded systems, development-production gap.

Análisis de Seguridad de Configuración SSL/TLS en Sistemas Embebidos Industriales Basados en Django: Evaluación a Nivel de Código

Resumen. Los sistemas de visión industrial basados en hardware embebido y frameworks web de desarrollo rápido presentan una superficie de ataque particular que ha recibido escasa atención en la literatura regional. En este trabajo analizamos la seguridad de PLABVision, un sistema de clasificación de productos en línea desarrollado mediante colaboración entre un instituto de investigación aplicada y una PYME manufacturera argentina, que opera sobre una plataforma NVIDIA Jetson Orin Nx con cámara industrial GigE y backend Django 4.2. La evaluación se realizó sobre el código de producción en un entorno de desarrollo equivalente, aplicando análisis estático de configuración y pruebas dinámicas con OWASP ZAP. El hallazgo principal es un gap de configuración SSL/TLS entre el entorno de desarrollo y el despliegue productivo: el sistema configura correctamente HTTPS en la capa de aplicación Django pero el servidor nginx que gestiona el tráfico real no tiene TLS habilitado, lo que resulta en transmisión de credenciales en texto claro. Esto fue demostrado mediante la interceptación de un request de autenticación con OWASP ZAP, capturando credenciales sin cifrado sobre HTTP. La evaluación se realizó en entorno de desarrollo local; la validación del vector en la red LAN industrial constituye trabajo futuro. Los resultados son aplicables a otros proyectos PYME que adopten stacks similares Django-nginx-hardware embebido, un patrón cada vez más frecuente en la industria argentina.

Palabras clave: configuración SSL/TLS, ciberseguridad PYME, seguridad web Django, sistemas embebidos industriales, brecha desarrollo-producción.

1 Introduccion

PLABVision es un sistema de visión artificial desarrollado mediante colaboración industria-academia para la clasificación automatizada de ampollas en línea de producción. Opera sobre una plataforma NVIDIA Jetson Orin Nx con cámara industrial GigE compatible con el estándar GenICam, e integra un módulo de control implementado como máquina de estados finitos (FSM) en Python. La interfaz de

usuario es una aplicación web Django 4.2 desplegada con Gunicorn y nginx, accesible desde navegador en la red local de planta. El sistema fue validado funcionalmente en laboratorio y campo (Ledesma et al., 2025).

El escenario genera una tensión que no siempre se documenta. La misma agilidad que hace atractivo a Django en proyectos de prototipado puede introducir configuraciones inseguras que persisten hasta el despliegue productivo. El desarrollador configura SSL/TLS en desarrollo, prueba que funciona, y luego el deployment en Docker con nginx reproduce la estructura funcional pero omite —o mal configura— la capa de transporte seguro. El resultado es un sistema que cree estar sirviendo HTTPS mientras las credenciales de sus operarios viajan en texto claro por la red de planta. Este tipo de exposición figura entre las fallas criptográficas más frecuentes en aplicaciones web (OWASP Foundation, 2021) y en variantes específicas para entornos IoT industriales (OWASP Foundation, 2018). En entornos de planta donde la red LAN puede estar físicamente accesible a múltiples actores, las consecuencias son concretas: captura de credenciales, acceso no autorizado a la interfaz de control, y potencialmente manipulación del proceso supervisado.

Las contribuciones de este trabajo son las siguientes. Primero, se identificó y demostró empíricamente un gap de configuración SSL/TLS entre el entorno de desarrollo Django (runserver_plus con certificados locales) y el despliegue productivo (nginx sin TLS), que resulta en transmisión de credenciales en texto claro. La demostración se realizó mediante intercepción activa con OWASP ZAP. Segundo, se analizó la cadena de configuraciones que produce esta vulnerabilidad —*SECURE_SSL_REDIRECT, SECURE_PROXY_SSL_HEADER* y la ausencia de TLS en nginx— y se propone un modelo de configuración seguro para despliegues Django-nginx en hardware embebido, en línea con las guías de evaluación establecidas (NIST, 2008). Además, el análisis identificó dos configuraciones inseguras menores: una SECRET_KEY con entropía insuficiente, corregidas durante la evaluación y un endpoint de monitoreo sin autenticación, para el cual se produjo su mitigación. Los resultados están orientados a equipos que desarrollen sistemas similares en contextos PYME.

El resto del trabajo se organiza de la siguiente manera. La Sección 2 revisa trabajos relacionados. La Sección 3 describe la metodología. La Sección 4 presenta los hallazgos. La Sección 5 discute las implicaciones. La Sección 6 concluye el trabajo

2 Trabajos Relacionados

Las configuraciones incorrectas de TLS son una de las categorías de vulnerabilidad más frecuentes en aplicaciones web desplegadas: certificados autofirmados sin validación, versiones obsoletas del protocolo, y —relevante para este trabajo— la desactivación efectiva de TLS en alguna capa intermedia del stack de despliegue. La OWASP Foundation clasifica las fallas criptográficas como el segundo riesgo más crítico en aplicaciones web (OWASP Foundation, 2021), categorizando

explícitamente la transmisión de datos sensibles en texto claro como una vulnerabilidad de alta severidad. Django incorpora protecciones nativas para este escenario: *SECURE_SSL_REDIRECT, SESSION_COOKIE_SECURE* y *CSRF_COOKIE_SECURE* están documentados en la guía oficial de deployment (Django Software Foundation, 2023). Sin embargo, la investigación académica sobre seguridad en Django se concentra en la capa de aplicación —vistas, autenticación, validación de inputs— y no en la interacción entre Django y la infraestructura que lo rodea. El mecanismo específico por el cual configuraciones correctamente definidas en el framework pueden volverse inefectivas cuando el reverse proxy no tiene TLS habilitado no ha recibido tratamiento sistemático en la literatura.

Desde la perspectiva de IoT industrial, Sivaraman et al. (2015, 2018) documentan patrones de tráfico inseguro y proponen mecanismos de segmentación de red para dispositivos conectados. Zarpelão et al. (2017) revisan enfoques de detección de intrusiones para entornos IoT con restricciones de cómputo. La OWASP Foundation sistematiza estos riesgos en su IoT Top 10, que incluye interfaces web inseguras y transferencia insegura de datos (OWASP Foundation, 2018). Sin embargo, estos trabajos se centran en el dispositivo embebido como nodo de red y no en las aplicaciones web que se ejecutan sobre él. La combinación específica de Django + hardware embebido industrial + evaluación sobre código de producción real es, hasta donde se ha podido identificar, escasamente documentada, y prácticamente ausente en el contexto latinoamericano. Este trabajo contribuye precisamente en esa intersección.

3 Metodología de Evaluación

3.1 Descripción del sistema

PLABVision es un sistema de visión industrial para clasificación automática de productos en línea. Corre sobre una NVIDIA Jetson Orin Nx conectada a una cámara industrial GigE compatible con GenICam. A nivel de software, utiliza Django 4.2 como framework web, Python 3.10 para la lógica de control, y OpenCV junto con Aravis para adquisición de imágenes.

El sistema se despliega con contenedores Docker, usando Nginx como reverse proxy de la aplicación Django. El código fuente analizado proviene del repositorio oficial del proyecto y corresponde al mismo utilizado en el entorno de producción. La validación funcional del sistema en el hardware objetivo fue confirmada a través de documentación técnica del proyecto (Ledesma et al., 2025).

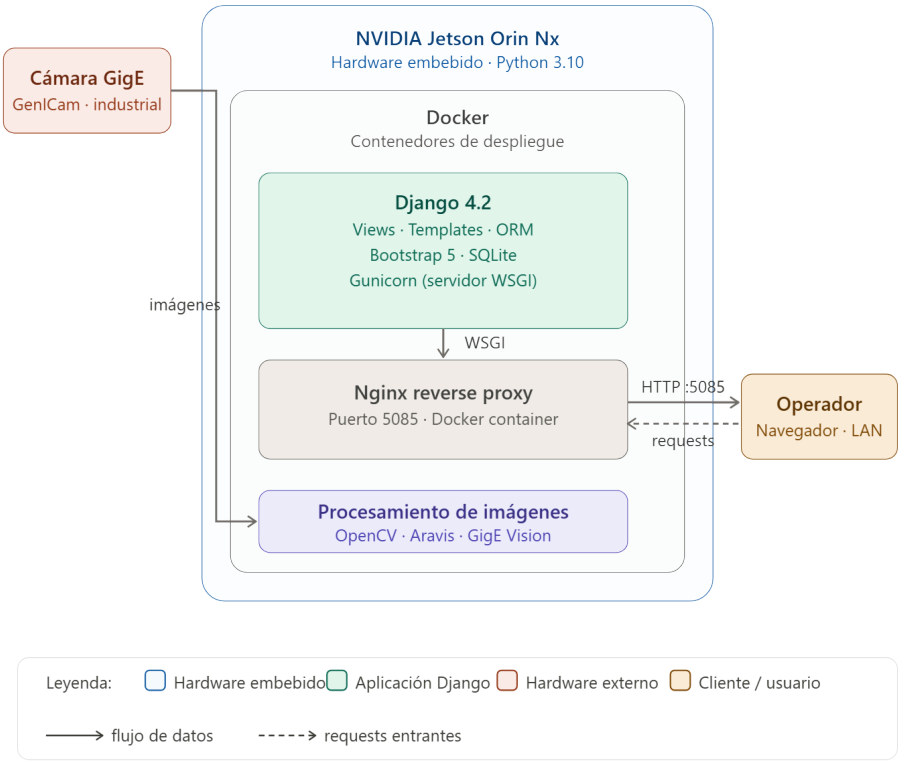

Fig. 1. El diagrama muestra el flujo completo: la cámara GigE industrial alimenta al Jetson Orin Nx (contenedor principal), donde dentro del entorno Docker conviven el stack Django 4.2/Gunicorn y el proxy Nginx, más la capa de procesamiento de imágenes con OpenCV y Aravis. El operador accede desde su navegador vía LAN al puerto 5085 de Nginx

3.2 Entorno de evaluación y herramientas

La evaluación se realizó sobre Ubuntu 22.04 LTS con Docker Engine 24.0+ y Docker Compose v2.x. Las versiones del stack fueron extraídas del archivo requirements.txt: Django 4.2.9, Python 3.10, Gunicorn 21.2.0, WhiteNoise 6.6.0, python-dotenv 1.0.1. El proxy inverso utilizó la imagen nginx:1.25-alpine.

Se emplearon herramientas estándar de análisis de seguridad (NIST, 2008). OWASP ZAP 2.14 fue configurado como proxy de interceptación para capturar tráfico HTTP/HTTPS. Bandit 1.7 se empleó para análisis estático del código Python. Además, se revisaron manualmente los archivos de configuración críticos: settings.py, nginx.conf y docker-compose.yml. Para la evaluación de severidad se utilizó CVSS v3.1, en concordancia con las categorías de riesgo definidas por OWASP (OWASP Foundation, 2021).

3.3 Procedimiento

El análisis se llevó a cabo en cuatro fases.

(a) Análisis estático: se revisaron manualmente los archivos de configuración de Django y del entorno de despliegue, identificando el gap SSL/TLS.

(b) Análisis automatizado: se ejecutó Bandit sobre el código fuente completo para detectar patrones inseguros; los resultados fueron filtrados para descartar falsos positivos.

(c) Demostración empírica: se configuró OWASP ZAP como proxy local y se interceptó tráfico hacia la aplicación en 127.0.0.1:8000, capturando credenciales de autenticación en texto claro.

(d) Hallazgos adicionales: el análisis reveló configuraciones inseguras secundarias, ambas corregidas durante la evaluación.

El análisis se centró en vulnerabilidades de configuración y despliegue —el desalineamiento entre configuraciones de seguridad definidas en el framework y su implementación real en producción— enfoque coherente con las metodologías recomendadas para este tipo de sistemas (NIST, 2008; OWASP Foundation, 2018).

La principal limitación es que la evaluación no se realizó sobre el dispositivo Jetson en producción, sino en un entorno de desarrollo sobre PC. La interceptación de credenciales se realizó en localhost (127.0.0.1:8000), no en la red LAN industrial. Las vulnerabilidades identificadas —en particular el gap SSL/TLS— derivan de la configuración del código y son independientes del hardware, por lo que pueden verificarse mediante análisis estático y reproducirse en entornos equivalentes. Dado que la evaluación fue realizada por el mismo equipo que desarrolló el sistema, se adoptaron metodologías estandarizadas y herramientas de análisis de terceros para mitigar el sesgo de confirmación, documentando todos los hallazgos relevantes incluyendo los que no derivaron en vulnerabilidades explotables.

4 Resultados

Análisis de configuración Django

El análisis del archivo settings.py reveló una configuración claramente orientada a forzar HTTPS. El sistema habilita *SECURE_SSL_REDIRECT = True, SESSION_COOKIE_SECURE = True y CSRF_COOKIE_SECURE = True*. Estas configuraciones reflejan una intención explícita de endurecer la superficie de ataque en la capa de aplicación.

```
1  # settings.py (fragmento relevante de seguridad HTTPS)
2
3  SECRET_KEY = os.environ.get('SECRET_KEY')  # [REDACTED]
4  DEBUG = str2bool(os.environ.get('DEBUG'))
5  ALLOWED_HOSTS = ['*']
6
7  CSRF_TRUSTED_ORIGINS = [
8     'http://localhost:5085',
9     'http://127.0.0.1:5085',
10     'http://192.168.X.X:8000'
11 ]
12
13 # --- CONFIGURACIÓN DE SEGURIDAD HTTPS ----
14 SECURE_SSL_REDIRECT = True      # Fuerza redirección a HTTPS
15 SESSION_COOKIE_SECURE = True        # Cookies solo por HTTPS
16 CSRF_COOKIE_SECURE = True           # CSRF solo por HTTPS
17 SECURE_PROXY_SSL_HEADER =('HTTP_X_FORWARDED_PROTO', 'https')
18 # -------------------------------
19
20 # SSL en desarrollo (certificados locales)
21 SSLSERVER_CERTIFICATE=BASE_DIR /
'certificates/.../localhost.crt'
22 SSLSERVER_PRIVATE_KEY = BASE_DIR /
'certificates/.../localhost.key'
```

Listing 1. Configuración de seguridad HTTPS en Django mediante directivas SECURE_*, indicando que la aplicación asume operación bajo TLS y delega la terminación SSL al reverse proxy.

El uso de *ALLOWED_HOSTS = ['*']* es una configuración permisiva que debilita los mecanismos de validación de host, aunque no constituye el vector de ataque principal identificado. El uso de *SECURE_PROXY_SSL_HEADER* indica que Django está desplegado detrás de un reverse proxy y confía en el header X-Forwarded-Proto para determinar si la conexión original fue HTTPS. Este esquema introduce una dependencia crítica: las protecciones activadas en settings.py solo son efectivas si el proxy está correctamente configurado para manejar TLS.

Análisis de configuración Nginx

La revisión de nginx.conf y docker-compose.yml identificó el problema central. Nginx no presenta ninguna directiva asociada a TLS: no hay bloques ssl_certificate ni ssl_certificate_key, ni configuraciones asociadas al puerto 443

```
1  # nginx.conf (configuración del reverse proxy)
2
3  upstream webapp {
4     server appseed_app:5005;
5  }
6
7  server {
```

```
8     listen 5085;                  # Solo HTTP (sin TLS)
9     server_name localhost;
10
11     location / {
12         proxy_pass http://webapp; # Backend Django
13         proxy_set_header Host $host;
14  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
15         # No se define X-Forwarded-Proto ni SSL
16     }
17
18     # AUSENCIA de:
19     # listen 443 ssl;
20     # ssl_certificate / ssl_certificate_key
21  }
```

Listing 2. Configuración de Nginx sin soporte TLS, exponiendo el servicio únicamente por HTTP y sin directivas de cifrado SSL.

```
1  # docker-compose.yml (fragmento de despliegue)
2
3  version: '3.8'
4
5  services:
6   appseed-app:
7     container_name: appseed_app
8     build: .
9     networks:
10       - web_network
11
12   nginx:
13     container_name: nginx
14     image: nginx:latest
15     ports:
16       - "5085:5085"   # Exposición directa por HTTP
17     volumes:
18       - ./nginx:/etc/nginx/conf.d
19     depends_on:
20       - appseed-app
21     networks:
22       - web_network
```

Listing 3. Configuración de despliegue Docker exponiendo el servicio Nginx vía HTTP en el puerto 5085

Aquí aparece el gap central. Django está configurado para operar bajo el supuesto de que las conexiones entrantes son HTTPS, pero Nginx no implementa TLS ni realiza terminación de cifrado. Como resultado, SECURE_SSL_REDIRECT no se aplica efectivamente porque el sistema no tiene un endpoint HTTPS al cual redirigir

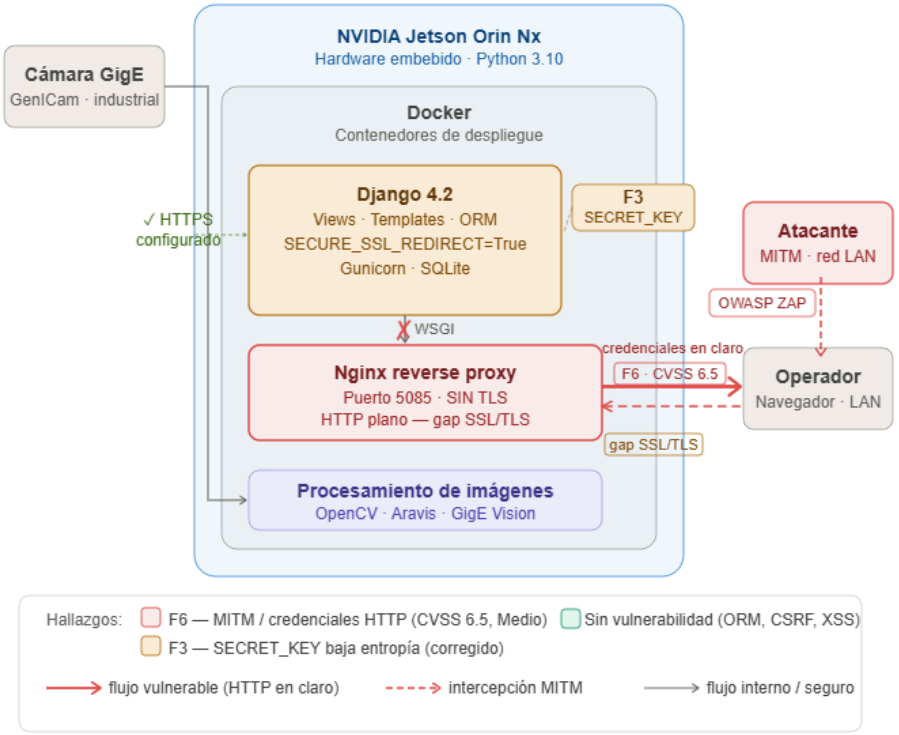

Fig. 2. Arquitectura PLABVision con hallazgos de seguridad identificados. En rojo: vector de ataque principal (F6, CVSS 6.5). En ámbar: hallazgo menor corregido (F3). La ✗ indica el desalineamiento entre la configuración HTTPS de Django y la ausencia de TLS en Nginx

Este desalineamiento no es un error de código en sí mismo, sino una inconsistencia entre capas del sistema. Su impacto es significativo: invalida completamente las protecciones de transporte definidas en la aplicación.

Demostración empírica con OWASP ZAP

Para validar el impacto práctico, se configuró OWASP ZAP como proxy local en 127.0.0.1:8081 y el navegador Firefox se ajustó para enrutar todo el tráfico a través de él. Se accedió a la interfaz de autenticación (/accounts/login/) y se realizó un login con credenciales de prueba. ZAP interceptó el POST generado por el formulario y mostró el contenido completo del cuerpo de la petición.

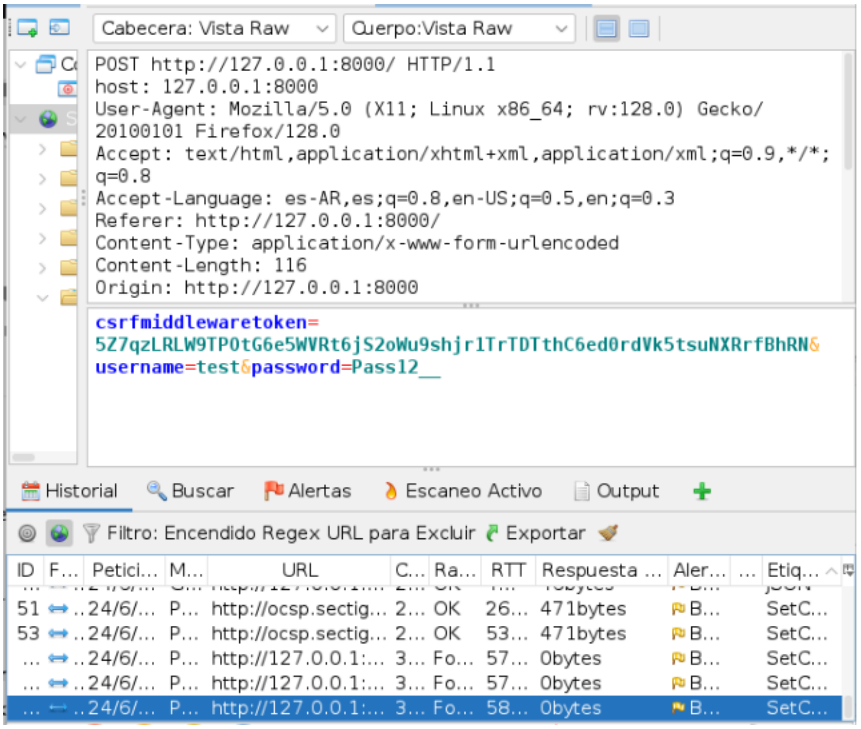

Fig. 3. Interceptación con OWASP ZAP del request POST al formulario de login. Las credenciales de autenticación se transmiten en texto claro sobre HTTP (① sin TLS). Se observan username y password visibles en el cuerpo del request (②).

Como se observa en la Figura 3, las credenciales fueron transmitidas en texto plano:

username=test
password=Pass12__

No se detectó ningún tipo de cifrado ni ofuscación. Aunque la solicitud incluía un token CSRF válido, este mecanismo no protege contra eavesdropping, ya que su función es prevenir falsificación de solicitudes, no garantizar confidencialidad. Un atacante con capacidad de interceptar tráfico en la misma subred —mediante ARP spoofing, acceso al switch, o man-in-the-middle— podría capturar las credenciales sin autenticación previa ni acceso privilegiado. La demostración se realizó en entorno local; la captura desde otra máquina en la LAN industrial real constituye trabajo futuro, pero la configuración del código de producción es suficiente para concluir la existencia de la vulnerabilidad. Este escenario corresponde directamente a una falla de tipo Cryptographic Failure, categorizada como OWASP A02:2021

Evaluación de severidad

El gap SSL/TLS fue evaluado con CVSS v3.1. El vector CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N produce un Base Score de 6.5 (Medio). El vector de ataque es Adjacent Network (AV:A): el sistema opera en una LAN industrial privada y no está expuesto directamente a Internet. La complejidad del ataque es baja (AC:L), no requiere privilegios previos (PR:N), y el impacto sobre la confidencialidad es alto (C:H).

La Tabla 1 resume los hallazgos identificados en este trabajo, junto con sus respectivos scores CVSS y estado actual (identificado, corregido o propuesto).

Hallazgo	CVSS	Severidad	Estado
Gap de configuracion SSL/TLS (Django vs Nginx)	6.5	Medio	Identificado
Generacion insegura de SECRET_KEY(random.choice)	4.3	Medio	Corregido
Endpoint de monitoreo sin autenticación	5.3	Medio	Propuesto

Además del gap SSL principal, el análisis identificó dos configuraciones secundarias. La SECRET_KEY era generada mediante random.choice() sobre caracteres ASCII, lo que no garantiza suficiente entropía para usos criptográficos; fue reemplazada por *secrets.token_urlsafe(50)*. Un endpoint de monitoreo (/pages/running/) resultó accesible sin autenticación, exponiendo métricas del sistema; la mitigación propuesta es incorporar el decorador *@login_required.*

5 Discusión

El gap SSL/TLS identificado en PLABVision ilustra una clase de vulnerabilidad que escapa, casi por definición, a las herramientas tradicionales de análisis de código. Django está correctamente configurado según su documentación oficial: *SECURE_SSL_REDIRECT* fuerza redirecciones a *HTTPS, SESSION_COOKIE_SECURE y CSRF_COOKIE_SECURE* protegen las cookies de sesión, y *SECURE_PROXY_SSL_HEADER* está definido para que el framework reconozca conexiones seguras detrás de un proxy. El problema no está en ninguna de esas líneas. Está en que Nginx, la capa que realmente recibe el tráfico de red, no tiene TLS configurado. Es una vulnerabilidad de deployment, no de programación, y esa distinción tiene consecuencias prácticas importantes.

El desalineamiento es silencioso: el sistema funciona correctamente, no genera errores visibles, y la vulnerabilidad solo se hace evidente cuando alguien intercepta el tráfico. En equipos pequeños existe la tendencia comprensible a tratar la seguridad de la aplicación y la del despliegue como el mismo problema. Lo que no siempre se considera es que runserver_plus con certificados locales y Nginx en producción son

entidades completamente diferentes: la primera es un servidor de desarrollo que Django controla directamente, la segunda es un proceso externo con su propia configuración TLS independiente de lo que settings.py indique.

La arquitectura evaluada no es particular de PLABVision. Django desplegado con Gunicorn detrás de Nginx en contenedores Docker es un patrón ampliamente adoptado, incluyendo sistemas sobre hardware embebido como Raspberry Pi o NVIDIA Jetson. Cualquier sistema con esta arquitectura que haya configurado TLS en el framework pero no en el reverse proxy presenta la misma exposición. En el contexto latinoamericano, donde la adopción de hardware embebido accesible para automatización industrial en PYMEs ha crecido sostenidamente, este patrón es especialmente relevante: los equipos que desarrollan estos sistemas suelen tener expertise en visión por computadora o desarrollo web, pero no necesariamente en seguridad de infraestructura. Los resultados de este trabajo sugieren que una auditoría de deployment —revisión de nginx.conf, docker-compose.yml y la cadena completa de configuración— debería ser parte del proceso de puesta en producción, con independencia de cuán correctamente esté configurada la capa de aplicación.

El aspecto más relevante para trabajo futuro es la validación del vector en condiciones reales de operación: repetir las pruebas directamente sobre el hardware embebido, capturando tráfico en la red de planta con Wireshark o tcpdump. También sería valioso extender el análisis a otros sistemas con arquitectura similar desarrollados en el marco de proyectos de colaboración industria-academia en la región, para determinar si el patrón identificado es idiosincrático de PLABVision o representa una tendencia más amplia. Finalmente, los hallazgos sugieren la utilidad de desarrollar una herramienta de verificación automatizada de gaps de deployment para proyectos Django, integrable en pipelines de CI/CD, que alerte cuando la configuración del reverse proxy no es coherente con las expectativas de seguridad definidas en settings.py.

6 Conclusiones

La evaluación de seguridad de PLABVision, un sistema de visión industrial embebido desarrollado mediante colaboración industria-academia sobre hardware NVIDIA Jetson Orin Nx, identificó un gap de configuración SSL/TLS de severidad media entre el entorno de desarrollo y el despliegue productivo. Django 4.2 está correctamente configurado para forzar HTTPS y proteger cookies de sesión, pero el servidor Nginx que gestiona el tráfico real en producción no tiene TLS habilitado, lo que resulta en transmisión de credenciales de autenticación en texto claro. Este vector fue demostrado empíricamente mediante la interceptación de un request POST con OWASP ZAP, obteniendo credenciales sin cifrado sobre HTTP, con una severidad evaluada en CVSS 6.5 (Medio).

El trabajo contribuye en dos planos. Técnicamente, documenta un vector de vulnerabilidad específico de la combinación Django-Nginx en entornos Dockerizados sobre hardware embebido industrial: una configuración donde el framework cree estar

operando sobre HTTPS mientras la capa de red subyacente no cifra el tráfico. Metodológicamente, aporta un caso de estudio sobre código de producción real en el contexto de una PYME argentina con respaldo de un instituto nacional de investigación aplicada, cubriendo una brecha identificada en la literatura sobre seguridad de deployment en sistemas Django embebidos. Los resultados son aplicables a cualquier sistema que comparta esta arquitectura —Django + Gunicorn + Nginx en Docker sobre Linux embebido— independientemente del dominio de aplicación.

El trabajo reconoce como limitación principal que la evaluación se realizó en un entorno de desarrollo sobre PC en lugar del hardware Jetson en producción. No obstante, las vulnerabilidades identificadas derivan de la configuración del código y del entorno de despliegue Docker-Nginx, que son propiedades independientes de la arquitectura de hardware donde se ejecuta el análisis. El gap SSL existe en el código que corre en producción; la evaluación en PC lo hace verificable y reproducible.

Más allá de PLABVision, los resultados señalan algo que trasciende el caso particular: la seguridad de un sistema no termina en el código. Las decisiones de deployment pueden introducir vulnerabilidades críticas aun cuando la aplicación esté correctamente implementada, y esa brecha merece atención sistemática en cualquier proceso de desarrollo de software industrial.

Referencias

Django Software Foundation. (2023). Security in Django. https://docs.djangoproject.com/en/4.2/topics/security/

First.org. (n.d.). Common Vulnerability Scoring System version 3.1: Specification document. https://www.first.org/cvss/v3.1/specification-document

Ledesma, F. H., Chuquimango, B., Vigna, A. M., Roldan, V. C. y Casal Bruno, S. (2025). Desarrollo de un prototipo de visión artificial para la clasificación de productos en línea. Jornadas Virtuales Transformación Digital e Industria 4.0, Instituto Nacional de Tecnología Industrial (INTI), San Martín, Buenos Aires, Argentina. https://icomunicacion.inti.gob.ar/2025/eventos/jornadas-TD-2025.pdf

NIST. (2008). Technical guide to information security testing and assessment (NIST SP 800-115). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-115

OWASP Foundation. (2018). OWASP Internet of Things Top 10. https://owasp.org/www-project-internet-of-things/

OWASP Foundation. (2021). OWASP Top 10:2021. https://owasp.org/Top10/

Sivaraman, V., Gharakheili, H. H., Vishwanath, A., Boreli, R. y Mehani, O. (2015). Network-level security and privacy control for smart-home IoT devices. En Proceedings of the 11th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) (pp. 163–167). IEEE. https://doi.org/10.1109/WiMOB.2015.7347956

Sivaraman, V., Gharakheili, H. H., Hegde, A., Ignjatic, A., Chew, C., Haas, J. y Nandakumar, R. (2018). Smart IoT devices in the home: Security and privacy implications. IEEE Technology and Society Magazine, 37(2), 71–79. https://doi.org/10.1109/MTS.2018.2826079

Zarpelão, B. B., Miani, R. S., Kawakani, C. T. y de Alvarenga, S. C. (2017). A survey of intrusion detection in Internet of Things. Journal of Network and Computer Applications, 84, 25–37. https://doi.org/10.1016/j.jnca.2017.02.002